Governors State University

OPUS Open Portal to University Scholarship

Fall 2023

Computer Vision for Robot Using AI

Sanjana Chandupatla

Computer vision for Robot Using AI

By

Sanjana Chandupatla
Vaagdevi College of Engineering
Bachelor of Technology in computer science, 2016

GRADUATE CAPSTONE SEMINAR PROJECT

Submitted in partial fulfillment of the requirements.

For the Degree of Master of Science,

With a Major in Computer Science

Governors State University
University Park, IL 60484

2023

ABSTRACT

Robotics is a multidisciplinary field that involves the design, construction, programming, and operation of robots. Robots are mechanical or electromechanical devices that are outfitted with sensors, processors, and actuators to carry out tasks autonomously or under the direction of a human. Robotics offers several benefits and uses in a variety of fields and industries. Robots can carry out repetitive activities with great accuracy and reliability. They can produce products with consistency in quality by performing operations with extreme accuracy.

Our anticipated approach merges traditional methodologies with forward-thinking innovations. Central to our design philosophy is the integration of tools such as Android Studio, GitHub, Git, and the Java programming language. We propose a robot with a constrained starting volume of 18x18x18 inches and a material flexibility of up to 0.25 inches. Integral to its design will be a web camera to enhance navigation capabilities, with the inclusion of April Tags for a comprehensive understanding of the game field's layout. The control system, a cornerstone of our design, is expected to feature an Android device interlinked with two team controllers, a driver station, and a Wi-Fi-enabled robot controller, with potential interfacing through the REV Robotics Expansion Hub or the REV Robotics Control Hub.

Our strategy envisions a robot capable of navigating the competition's multifaceted challenges, including a 30-second autonomous period, a two-minute driver-operated phase, and a climactic 30-second endgame. For robot manipulation learning, we provide a vision-based architectural search method that identifies relationships between high-dimensional visual inputs and low-dimensional action inputs. Our method automatically creates structures as it learns the task, coming up with fresh ways to associate and attend picture feature representations with actions and features from earlier levels. Comparing the obtained new architectures to a current, high performing baseline, they show superior task success rates, sometimes by a significant margin.

Table of Content

1 Project Description

The world as we know it today has been revolutionized by the advancements in the field of robotics. A remarkable blend of engineering and computer science, robotics offers us a glimpse into a future where machines can replicate or even surpass human capabilities.

This is our robotics competition. The main aim of FTC is to engage students in science, technology, engineering, and mathematics (STEM) fields by challenging them to design, build, program, and operate robots to compete in a themed game. FTC introduces a new game with specific rules, challenges, and objectives. The game for each season is designed to be both competitive and cooperative, encouraging teamwork, problem-solving, and innovation. The goals of the game is to involve tasks that robots must perform autonomously and/or under driver control.

The game is introduced with specific game elements, scoring mechanisms, and rules. The game is designed to be challenging and requires robots to perform a variety of tasks. Our Red Team design, build, and program a robot to complete the tasks outlined in the game and compete with other teams. The robot must be robust, versatile, and able to operate autonomously during certain parts of the game.

Our Red Team project's primary goal is to create a sophisticated control and software system for a robot, enabling it to navigate autonomously. This entails empowering the robot with the ability to autonomously calculate distances, identify April tags using a webcam, and precisely maneuver in the desired direction, thereby acquiring accurate spatial awareness within the game field. Once the assigned tasks are accomplished, the robot should seamlessly relocate itself to the specified destination, ensuring it concludes its operations precisely where required.

1.1 Competitive Information

Our Team 2 represents Red Alliance. Our team is going to compete with Team 1 which is Blue Alliance. In a match, there are different parts that together last for two minutes and thirty seconds (2:30). First, there's a thirty-second Autonomous Period, where the robot works on its own. Then comes a two-minute Driver-Controlled Period, where the team controls the robot. The last thirty seconds of this period is called the End Game. Between the time the robot works on its own and the time the team takes control, there's an eight-second break. This break lets teams grab the controllers and change the robot's program if needed. Once the Game is done the Robots earn points for their Alliance based on the criteria. A team consists of up to two driver operators, a human player, robot and a coach. Each match is played with four randomly selected teams two per alliance. Each alliance is allowed with one human player and our team is of RED alliance where we compete with BLUE alliance team. Robots are built from materials that are mentioned in the game manual. The primary element of scoring is the plastic hexagonal shaped pixel three inches across and by one half inch thick. We can construct custom games and scoring elements including team procs. The game is played on a 12 foot square playing field with a foam title floor and one-foot-highwalls. In back of the field are the backdrops one for each alliance beneath the backdrops are the taped off backstage areas.In the front corners are tapped off wings in each quadrant of the field are the three separate spike marks. Just inside the front wall are the white pixel location strips and outside of the front wall there are three taped off landing zones. April tags are located on the field wall and both the backdrops to aid navigation. Blue and red alliance stations are on the left and right sides of the field and in front of those are the red and blue team human player stations.
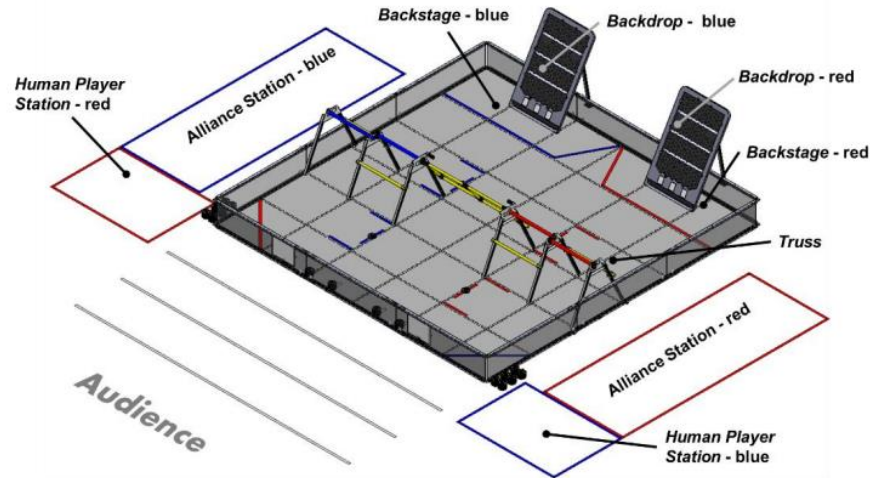
Figure 1.1.1 Overview of GameField

Our Red Alliance team-2 members are Ragini Malladi, Sanjana Chandupatla, Bhargavi Ajjarapu, Yamini Pupalla, Nazma Shaik. We have named our Robot as "**ChronoMech**" which means "chronos" (time) and "mech" (mechanism), indicating a robot with precise time-related functionalities to meet the competition requirements.The control center for the robot is called "**LBKX**," and you can easily find and connect to it using Wi-Fi.

1.2 Relationship to Other Applications/Projects

Team One initiated the process by accessing the relevant details about the competition on the FTC website. The initial action involved examining the available information and acquiring the following applications through downloads
GitHub Desktop Installation:
1. First we need to visit GitHub Desktop website at https://desktop.github.com/.
2. Click on the "Download for Windows" button to download the GitHub Desktop installer.
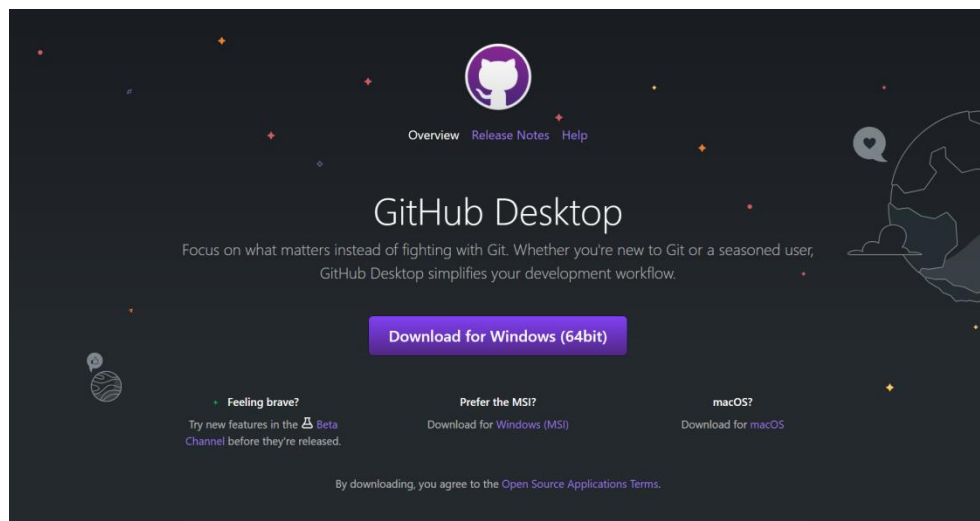

Figure 1.2.1 Installing GitHub Desktop

.

3. The GitHub Desktop Setup Wizard will appear. Click on "Next" to proceed.

4. Read the license agreement, and if you agree, select the checkbox indicating acceptance. Then, click "Next."

5. You can choose the installation options such as the destination folder for the installation. The default settings are usually fine for most users. Click "Install" to proceed.

6. The installer will now download and install GitHub Desktop. Wait for the process to complete.

7. Once the installation is finished, you can choose to launch GitHub Desktop immediately by leaving the checkbox selected. Click "Finish."

8. GitHub Desktop will open, and you will be prompted to sign in to your GitHub account. If you don't have an account, you can create one.
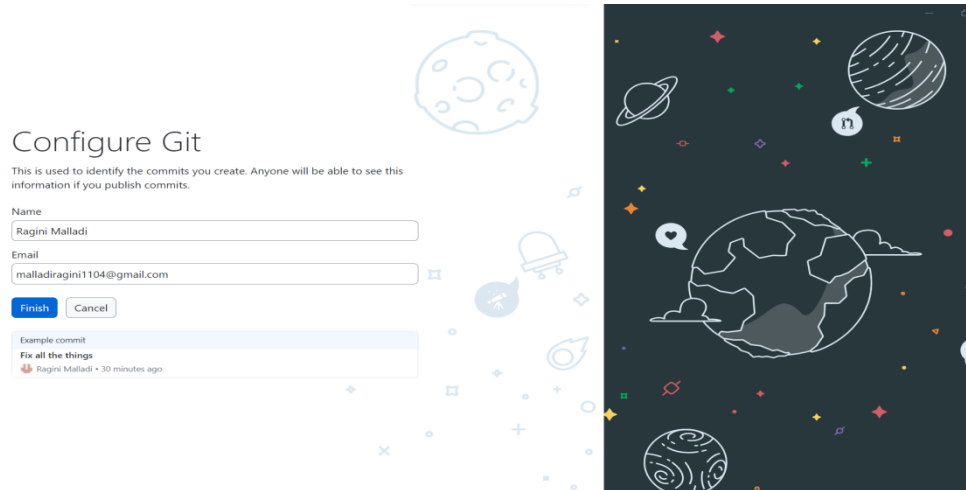

Figure 1.2.2 Setup an GitHub account or Sign In

9. Follow the on-screen instructions to configure GitHub Desktop settings, including your name, email, and default branch settings.

10. After completing the setup, you can start using GitHub Desktop to clone repositories, create branches, commit changes, and manage your GitHub projects.
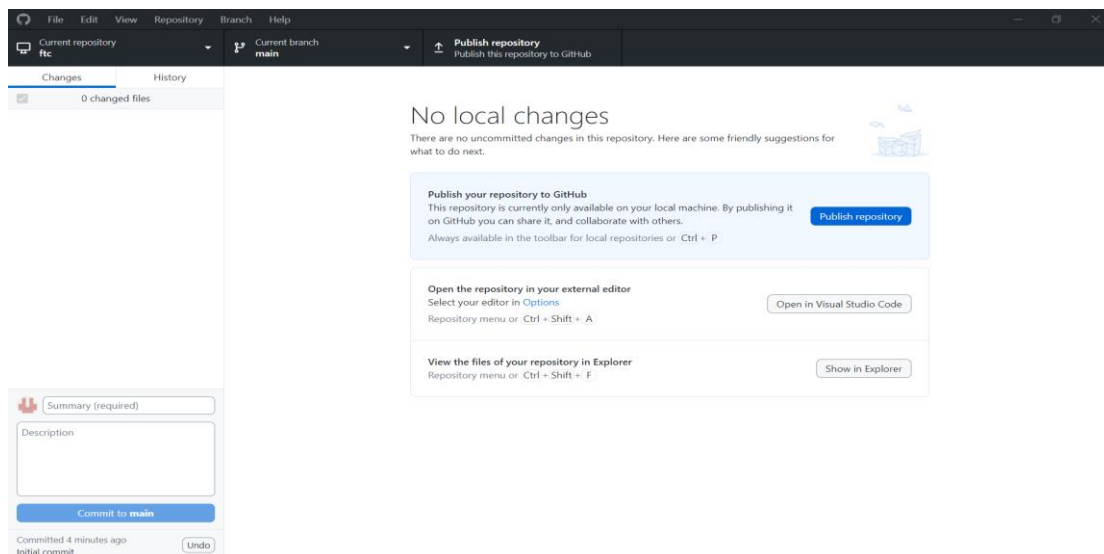

Figure 1.2.3 Github Account is Created
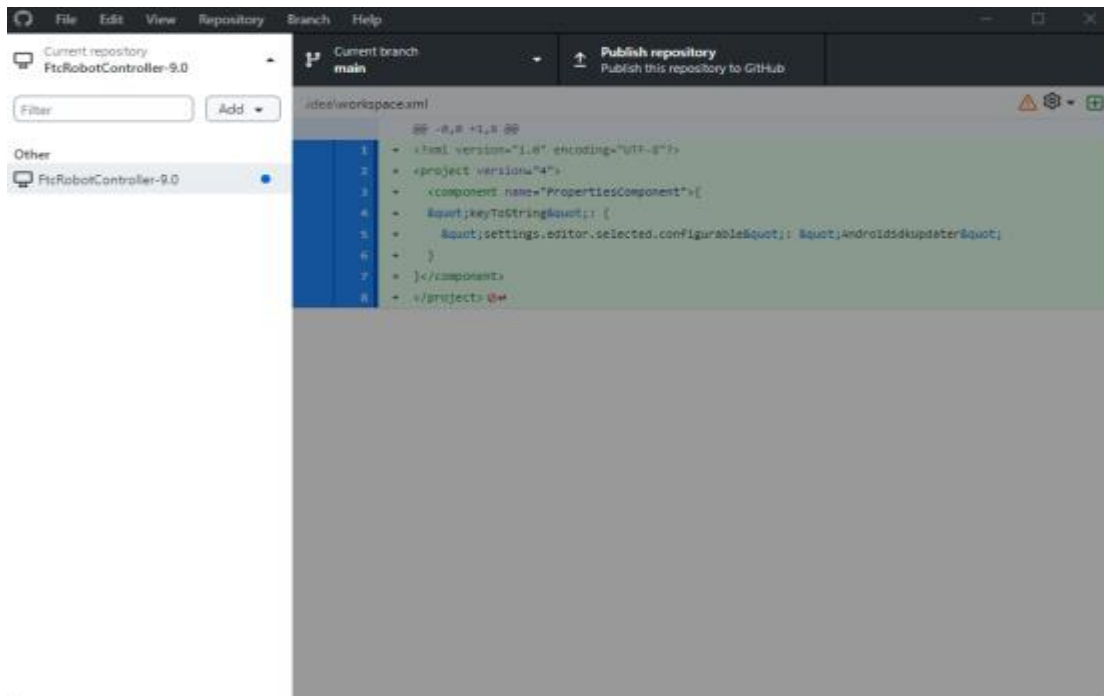
Figure 1.2.4 Publish Respository

Andriod Studio Installation:

1. Visit the official Android Studio website: https://developer.android.com/studio.
Click on the "Download Android Studio" button.
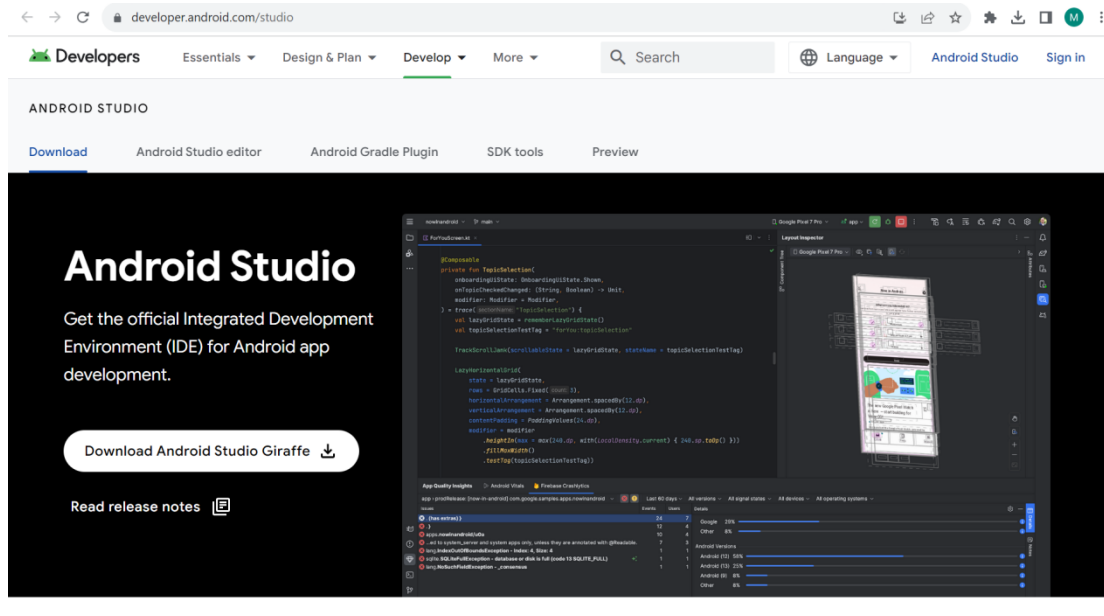
Figure 1.2.5 Andriod Studio Installation

2. Once the download is complete, run the installer.
3. You'll see a welcome screen. Click "Next" to continue.
4. Select the components you want to install. The default selections are usually sufficient. Click "Next."
5. Choose the installation location or use the default location. Click "Next."
6. Choose the Start Menu folder or use the default. Click "Install."
7. The installation process will begin. It may take some time to download and install the necessary files.
8. Once the installation is complete, click "Next."

9. The Android Studio Setup Wizard will launch. Click "Next."
10. Choose the type of setup you want for the Android Studio and the Android Virtual Device (AVD). Click "Next."
11. Review your settings and click "Finish" to complete the installation.
12. Once the download is complete, click "Finish" to exit the setup wizard.
Open Android Studio. The first time you open it, you may be asked to import settings or configure updates. Follow the on-screen instructions.

Figure 1.2.6 Andriod Studio
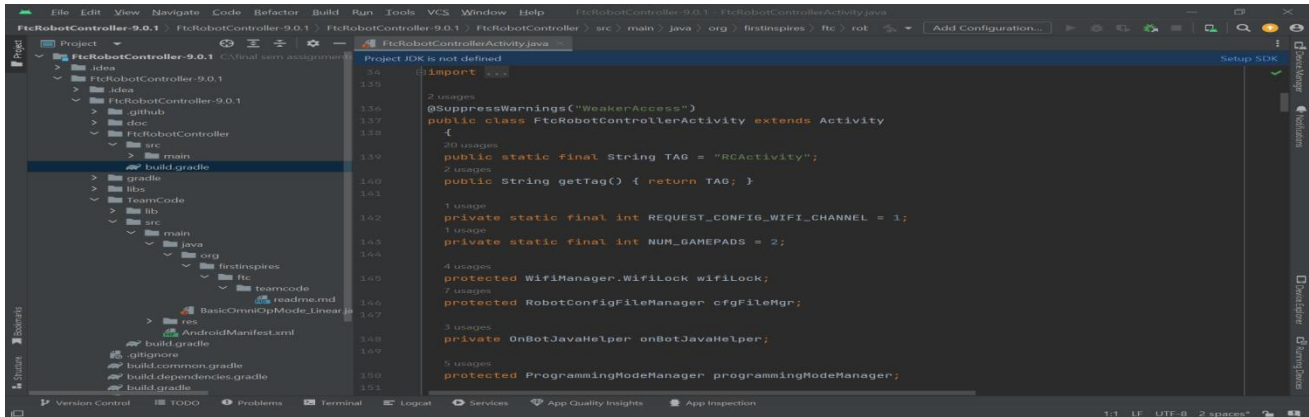
1.3 Assumptions and Dependencies

The FTC competition rules in autonomous games are the primary determinant of the robot's performance during the Autonomous Period and our team's strategy. The goal of the game is to complete the achievements to gain as many points as possible. Throughout the Autonomous Period, robot activities accrue points: Heading backstage to their Alliance. Arranging pixel art in their alliance backstage area or on their backdrop. Finding the Randomization Object on the Spike Mark that was chosen at random. Robots that plant Pixels get points in the following ways: Each pixel on the Alliance Backdrop's recessed scoring area is worth five points. Each pixel in the Alliance Backstage is worth three points.

Robots can acquire points in the driver-controlled Period by: Setting up Pixels in their Alliance Backstage or on their Alliance Backdrop. Pixels that receive points for crossing their alliance backdrop's set line.
Apart from the Driver-Controlled Period Scoring tasks mentioned above, Alliances can also obtain points by Sustaining Robots from their Alliance Rigging. Robots parking at the Alliance backstage area.
By navigating to areas of the playing field and engaging with scoring elements, robots can gain points for their alliance. Robots receive five points apiece when they park in the backstage area for the matching Alliance. Depending on where the Spike Mark was placed during randomization, there are two different tasks. Points for Randomization Tasks can only be obtained by a robot using its own Pre-Loaded Pixels.
Points are awarded to robots that place Pixels: Each Pixel that is in the recessed Scoring region of their Alliance Backdrop is worth three points. One point is awarded for each pixel in their Alliance Backstage.
When Scored Pixels on a backdrop extend along a horizontal Set Line, alliances receive ten points. Regardless of how many Pixels cross a set line, a vertical crossing awards one set bonus. An Alliance may receive a maximum of thirty points as a Set Bonus.

1.4 Future Enhancements

Greater use of sensors: Sensors could be used to give robots a better understanding of their surroundings, which could help them to perform tasks more effectively. For example, sensors could be used to detect objects, measure distances, and track movement.

More powerful motors and batteries: More powerful motors and batteries would allow robots to move faster and for longer periods of time. This would make them more versatile and capable of performing a wider range of tasks.

More advanced software: More advanced software would allow robots to perform more complex tasks and to make decisions on their own. For example, software could be developed that allows robots to recognize objects, understand language, and plan and execute complex tasks.

More affordable components: More affordable components would make it easier for teams to participate in the REV FTC program. This would help to increase the diversity of the teams that participate in the program and to encourage more students to get involved in robotics.

1.5 Definitions and Acronyms

FTC: First Tech Challenge

TeleOp's: TeleOps, short for TeleOperation, refers to a mode of operation in robotics where a human operator controls a robot remotely. Unlike autonomous operation, where the robot functions based on pre-programmed instructions, TeleOperation involves real-time control and decision-making by a human operator.

Autonomous: An autonomous robot is designed to perceive its environment, process information, and execute actions based on its programming and sensor inputs, without requiring continuous guidance from a human operator.

Android Studio – Android Studio is an integrated development environment (IDE) used for creating Android applications. It provides tools for coding, testing, and debugging Android apps in a user-friendly environment.

OpMode: In robotics, an OpMode (Operational Mode) is a program or code segment defining the behavior and actions of a robot during a specific phase of operation, such as autonomous or teleoperated mode. OpModes are crucial in defining how a robot responds to various inputs and tasks during a competition or operation.

2 Project Technical Description

2.1 Application Architecture

The match has an *autonomous* phase and a *driver-controlled* or *"tele-operated"* phase. In the autonomous phase of a match the robot operates without any human input or control. In the driver-controlled phase, the robot can receive input from up to two human drivers.

Our team leverages Android Studio, which enables us to write Java code for our competition robots. The control system is divided into two phases: the Robot Controller, which is mounted on the robot and serves as its brain, can use either the REV Robotics Expansion Hub or the integrated REV Robotics Control Hub with an Android device. The Driver Station, a second Android smartphone stationed with the team drivers and equipped with one or two gamepads, issues commands to the Robot Controller, which runs the corresponding apps. Op modes (Operational modes) are customized computer programs built with the Android Studio IDE to specify the competition robot's behavior.

Figure 2.1.1 Diver Station

The REV Robotics control Hub is the electronic input/output (or "I/O") module that lets the Robot Controller talk to the robot's motors, servos, and sensors. The Robot Controller communicates with the Expansion Hub through a serial connection. For the situation where a GamePad is used as the Robot Controller, a USB cable is used to establish the serial connection. For the situation where a REV Robotics Control Hub is used, an internal serial connection exists between the built-in GamePad and the Expansion Hub.The Expansion center, which is powered by a 12V battery, is one of the key components of our control center. This battery not only powers the Expansion Hub, but also the motors, servos, and sensors that are linked to it. For optimal operation, an independent battery powers the Gamepad when the Expansion Hub serves as the Robot Controller. When a REV Robotics Control Hub is used as the Robot Controller, the primary 12V battery powers both the Control Hub and its accompanying Expansion Hub. This ensures a unified power supply for smooth operation. The Expansion Hub, which oversees administering the Driver Station App, is at the heart of our Driver Station. We have the option of using a REV Driver Hub or a comparable solution to suit Gamepad requirements, giving a comprehensive and efficient control configuration for our robotic system.
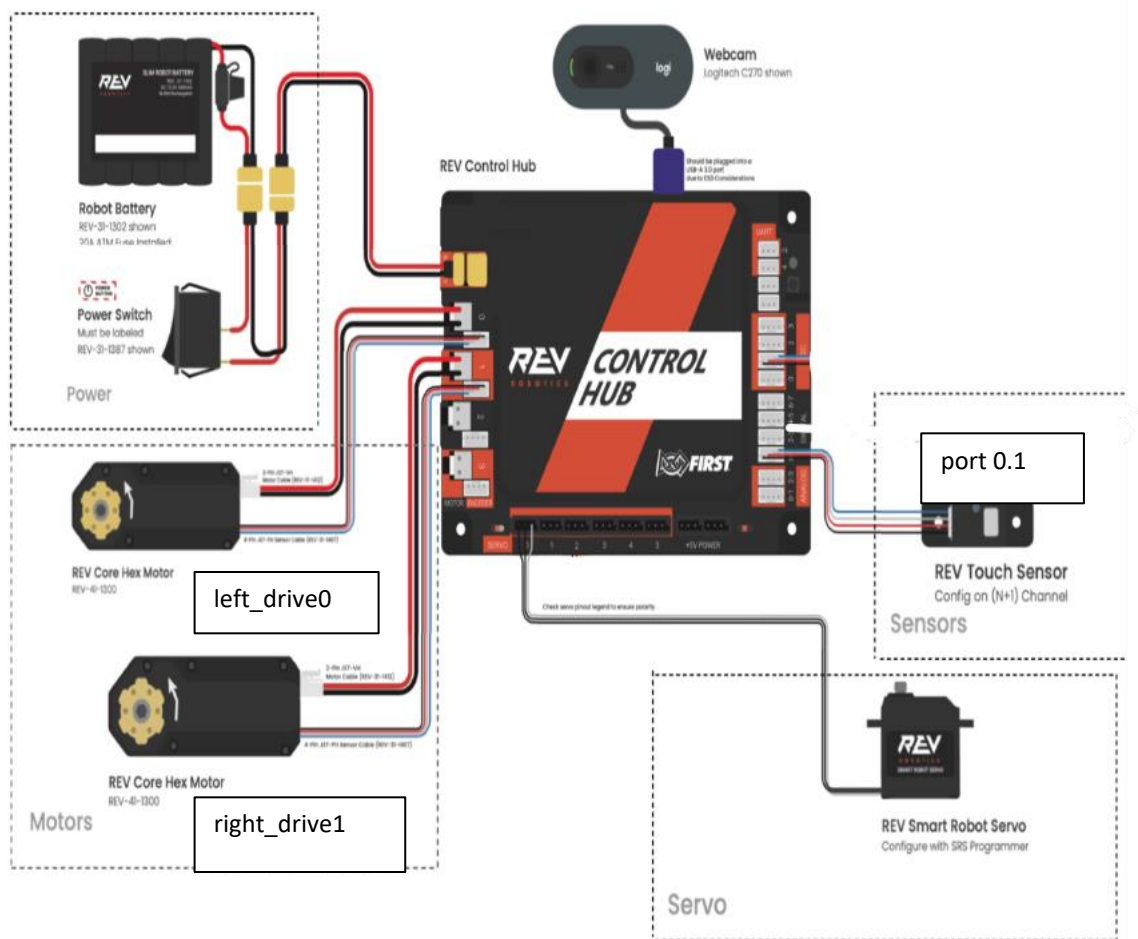
Figure 2.1.2 Overview of our Control Hub

2.2 Application Information flows:

During the autonomous portion of robotics contests, robots use preprogrammed instructions to complete tasks and roam the field, receiving points for fulfilling particular goals. The team with the highest final score receives an autonomous bonus, emphasizing the significance of demonstrating robot skills and improvements in autonomous systems technology.

Transition from Autonomous to Driver-Controlled Period: Robots will stay in a passive state when the Autonomous Period ends. Field staff will not enter the field or come into contact with any robots when the field is transitioning from autonomous to driver controlled. Drive Teams will receive visual and auditory indicators from the scoring system display when it's time to pick up their driver stations. Five seconds will be given to Drive Teams to set up and organize their Driver Station. Following a countdown of five seconds, the Driver Controlled Period of the Match will start.

The Autonomous Period of the match begins with robots that can only be controlled by preprogrammed commands. Using the Driver Station or any other tool, we are not permitted to direct Robot behavior when it is in the Autonomous Period. During the Autonomous Period, the Driver Station is positioned in a hands-off manner to make it clear that robots are not under human control. Allowing Drive Teams to use the "start" command on the Driver Station touch screen to activate their robot is the lone exemption. Teams are required to use the included 30-second timer. There's a countdown and then the Autonomous

Period starts. Using our Driver Station, we instruct the robot to operate in the Autonomous Op Mode that was chosen during the Pre-Match setup.

Drive Teams have a few seconds after the Autonomous Period ends to get their Driver Stations ready for the start of the 120-second Driver-Controlled Period. Drive Teams hit their Driver Station start button to commence playing the match when the countdown word "go" sounds, signalling the beginning of the Driver-Controlled Period.
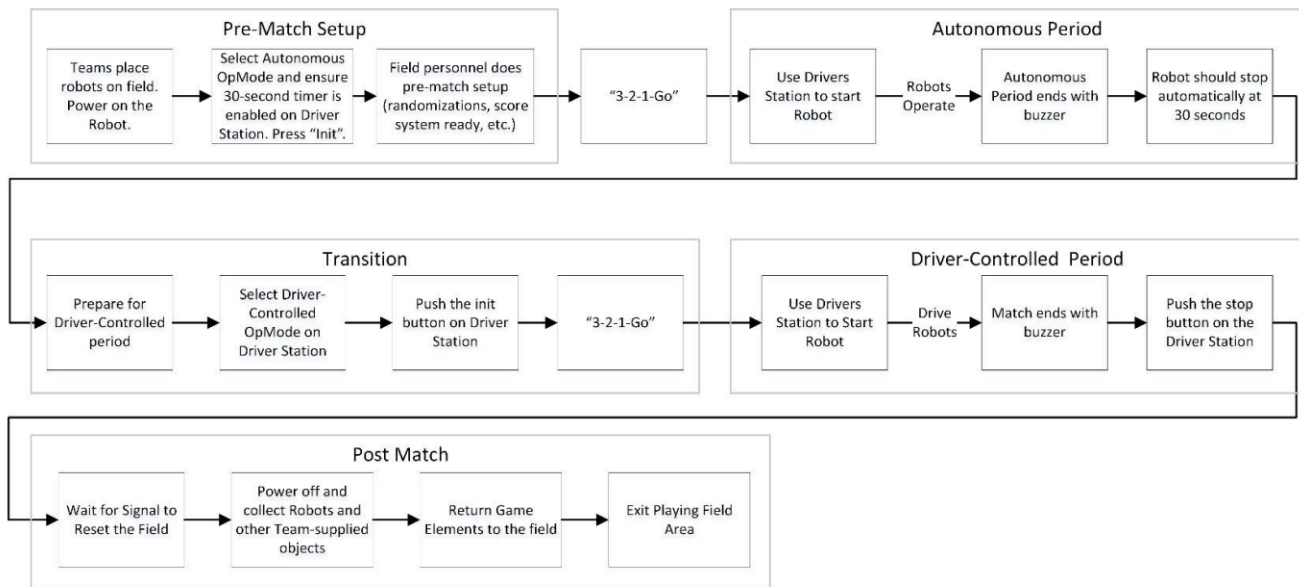

Figure 2.2.1 Application Flow

2.3 Interactions with other Projects

Simulation software: Before building their robots, teams can design and test them using simulation software. In addition to helping to avoid issues later, this can assist in saving time and resources.

Data analysis tools: To gather and examine data from their robots, teams might make use of data analysis tools. The robots' performance can be enhanced by using this data, which can also be utilized to find new areas for innovation.

Develop partnerships with other organizations: develop partnerships with other organizations, such as simulation software companies, data analysis companies.

Provide training and resources: provide training and resources to help teams use these new technologies.

2.4 Capabilities

Engaging in this challenge provides teams with a valuable opportunity to learn about robotics and actively participate in STEM. The program aids in the development of skills essential for future careers, offering an enjoyable, challenging, and rewarding experience.

Teams require a diverse set of abilities, including technical expertise in computer science, electrical engineering, and mechanical engineering, to design, construct, and program robots. They must also possess problem-solving skills to address issues that arise during the competition, covering team dynamics, game-related challenges, and robot functionality.

Furthermore, effective teamwork skills, encompassing decision-making, dispute resolution, and idea sharing, are crucial. Teams need to demonstrate adept time management, involving organization, planning, and meeting deadlines. Resourcefulness is another vital skill, enabling teams to think creatively and independently resolve challenges. Lastly, teams must cultivate creativity and innovation to tackle difficulties in unique and inventive ways.

2.5   Risk Assessment and Management

The program entails certain risks, including safety concerns if robots are not operated correctly. Team members must prioritize wearing proper safety gear and adhering to safety regulations when working with robots to mitigate these risks.

Careless use of robots by team members may lead to injuries such as cuts, burns, or shocks, emphasizing the importance of cautious operation. Improper use of robots in the FTC program may also result in property damage, underscoring the need for careful handling within a secure setting.

Electrical shock is a potential risk if robots are not operated correctly, emphasizing the importance of following safety protocols and operating robots in a secure environment. Additionally, improper use of robots may lead to data loss, highlighting the need for students to routinely back up their data.

Teams face reputational damage risks if they behave impolitely or fail to comply with competition regulations. To manage these risks, students should prioritize data backup practices, adhere to professional conduct rules, and ensure that all participants receive comprehensive safety training covering the associated risks and preventive measures.

3   Project Requirements

3.1   Identification of Requirements

The specifications for the project cover a wide range of factors, including robot design, building, programming, and team documentation. Teams must register via the official FIRST site, giving correct details about mentors, team members, and contact information. The creation of a robot that complies with the requirements and limitations given in the Game Manual for the competition season is the central task of the project. This entails building a robot that adheres to weight and size restrictions, incorporating a dependable electrical system, and putting autonomous and teleoperate control programmer into place. Teams are pushed to go outside the box and come up with novel features or mechanics that make their robots stand out.
Components:
1. Path Planning with Encoders
Path planning is crucial for the robot to navigate from its current position to the parking spot without collisions. Encoders provide valuable information about the robot's movement, helping to calculate distances and angles accurately.
2. Collision Avoidance
To prevent collisions with other robots, the system needs to constantly check the distance, position, and angle relative to nearby obstacles. This information is used to adjust the robot's path dynamically, ensuring it avoids any potential collisions.

3. Initialization and Movement

Drive Speed Values: During initialization, set appropriate drive speed values to ensure smooth and controlled movement.

The robot starts from tile E and move forward until tile 5 -Drives forward for 6.0 units at a speed defined by **DRIVE_SPEED** from here it takes a step back and turn slightly to right by moving 1.5 units in one direction and -1.5 units in the opposite direction, using a speed defined by **TURN_SPEED** and wait for moves forward from tile 5E all the way to tile 5A without any collisions

Reverse Values: Determine reverse values for backing up, as the robot might need to adjust its position or retreat from obstacles.

Sleep Time: Define sleep time between movements to allow the robot to stabilize and accurately perceive its environment.

Pauses the execution for 3000 milliseconds (or 3 seconds). Drives forward for 5.0 units at a speed defined by **DRIVE_SPEED**.

4. Parking Position Selection

The robot needs to intelligently choose a parking position, either on the left or right, to avoid collisions with other robots. Implement decision-making logic based on the surrounding environment.

5. April Tag Identification

To identify the parking spot, use a camera and implement a code to recognize April Tags. The code should be capable of determining the tag's position—whether it is in the middle, left, or right.

3.2 Operations, Administration, Maintenance and Provisioning (OAM&P)

OAM&P in the context of robots refers to the comprehensive set of activities and processes involved in managing and ensuring the efficient operation, administration, maintenance, and provisioning of robotic systems. This encompasses the following key elements:

Operations: Involves overseeing the day-to-day functioning of robotic systems, ensuring they perform tasks as intended, and monitoring their operational status. This includes managing tasks, coordinating activities, and optimizing operational efficiency

Administration: Encompasses the organizational and managerial aspects of robotic systems. It involves tasks such as resource allocation, system configuration, user access control, and overall governance to ensure smooth functioning and coordination within the robotic ecosystem.

Maintenance: Focuses on preserving the health and functionality of robotic systems over time. This includes preventive maintenance, troubleshooting, and addressing issues to minimize downtime. Regular maintenance activities aim to enhance reliability and extend the lifespan of the robotic hardware and software components.

Provisioning: Involves the allocation and management of resources needed for the operation of robotic systems. This includes the provisioning of hardware, software, network resources, and any other necessary components to support the robot's tasks and functions.

4   *Project Design Description*

Our robot's mechanical architecture is based on a study chassis made of lightweight but strong components that balances strength and agility in competitive play. A modular architecture built into the design makes it simple to customize and quickly adapt to different gaming scenarios. An essential component of our design is the manipulator mechanism, which has an adjustable arm and a precision gripper system for handling various game pieces with adaptability. The control Console of Robot Wi-Fi Name – lbkx assigned to Team One includes the following hardware. A REV(REV-11-1232) is operated by pulling power from a slim 12V battery. The battery is then connected to the connector on the control hub. The control hub had an LED which notifies us of any connections. When the LED is green the hub is ready for connection, LED will turn blue every 5 seconds indicating the hub is healthy. The hub comes with an orange USB and USB-C wire that is used to connect the hub with an A.S. programmed code file. The default network and password for the REV Control Hubs are included. Teams are advised by REV Robotics to use a 5 GHz channel for robot communication during competition. The REV control hub is a single gadget that combines an Android device with an expansion hub. The battery features an integrated 20A fuse for safety. Using a mechanical switch, the electricity is turned on and off. The hub is in the motor part and contains four ports, numbered 0 through 3, each of which connects to a 4-pin JST-PH type connector that powers the motor. When two quadrature encoders are used in tandem with a motor next to them, a 4-pin JST-PH type connector is utilized.
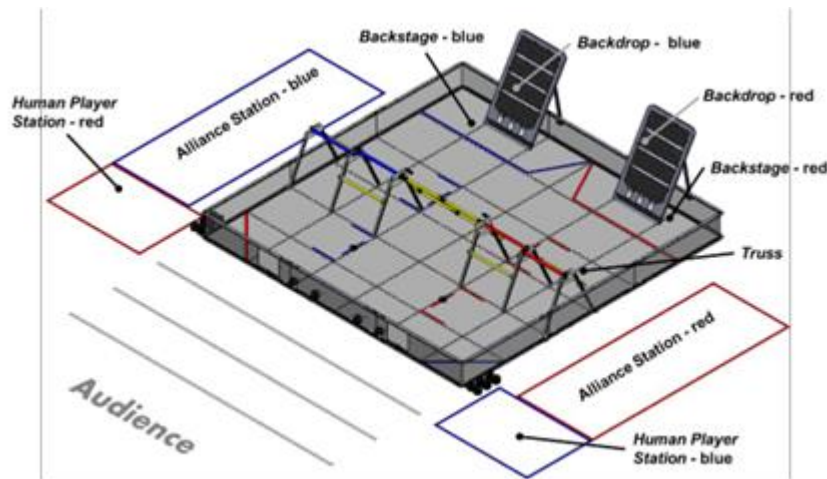

Figure 4.1 View of playing field

The connector is inserted into the control hub's encoder label. Servo Hub includes six built-in servo ports (servo is stated in orange on the hub, followed by input 0-5). The ground pin is located on the left side of the servo port, and they take conventional 3-wire header-style connectors. The servo on Robot (insert name) is set to input 0. On the digital connector with the identification 0-1, there is a Touch Sensor. The sensor contains two digital I/O pins for a total of eight digital I/O pins, four independent digital input/output (I/O) ports, and a REV Robotics Touch sensor connected to one of the digital I/O ports.

Project Application Execution

This project will be carried out in two modes: autonomous and teleop. These modes reflect many phases of operation, each of which serves a specific role in regulating and directing the robotic system. The autonomous mode consists of pre-programmed operations that allow the robot to complete tasks without direct human involvement. Teleop mode, on the other hand, relies on real-time control via a user interface, generally a game

controller or a similar device, allowing operators to actively steer and manipulate the robot. The incorporation of different modes allows flexibility and adaptation in tackling the project's numerous duties and obstacles.

Autonomous Mode Execution: The Autonomous_Opcode.java class, which is dedicated to allowing autonomous testing, is an essential component of our robotics project. The autonomous phase of robotics involves the robot doing pre-programmed tasks or responding to conditions without direct operator input. This class provides a structured environment in which developers can design, implement, and test autonomous behaviors.

Choosing Autonomous Operation Modes: We strategically selected which Autonomous Op Mode to use before the battle based on the game's objectives. This decision had an impact on the robot's behavior during the autonomous period.

Initial Position of the Robot: As stated in the game guide, the RED Alliance picked tile E6 as our robot's starting point. For the robot to recognize the April tag that was attached to it, we made sure it was precisely in the correct tile E5 area, made any necessary adjustments, and aligned it.

April Tag Detection and Pixel Positioning: During the Autonomous Period, our robot utilized the webcam to locate the April tag that was connected to the intended Pixel placement position. The robot then precisely moved to avoid obstacles and lined up the Pixel with the designated line.

Detecting the April Tag Using Web Camera: A popular camera-based technology is AprilTag, a scanned image like a QR Code. Its effectiveness and quick set-up on custom Signal Sleeves led to wide adoption, especially those programming in Java. Those POWERPLAY teams, including those using FTC Blocks, learned how to use several resources: AprilTag: an open-source technology for evaluating formatted images

EasyOpenCV: a FIRST Tech Challenge-optimized interface with OpenCV, an image processing library
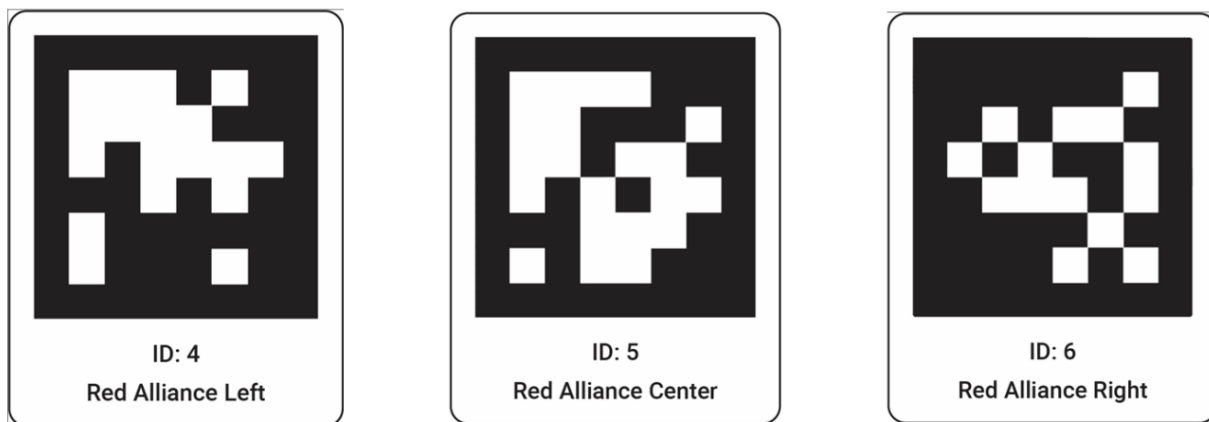
Sample April Tags:

Figure 4.2 April Tags

As the Red Team, we will be equipped with Red Alliance April Tags designed for detection within the game. These special April Tags serve as visual markers or identifiers deliberately placed to assist our robot in recognizing and traversing the gaming field. Using these Red Alliance April Tags improves our robot's capacity to perceive its environment and make smart judgments during games. This visual signal system contributes to the overall strategy and efficacy of our team's approach in the game.

Creating the Path and Dodging Obstacles: Because we had designed the robot to follow a predefined path, it was able to navigate the field and do not run into any obstacles. Precise and efficient navigation was critical to the robot's ability to do independent tasks. By adding these details to our project report, you may give readers a clear and thorough understanding of how our team addressed the autonomous mode of the competition and finished the tasks that needed to be done. It demonstrates our team's technical expertise and strategic insight.

According to the item that is present in the field.

a.   When the robot is situated in the center of the second tile **(E5)**, as seen in the diagram below.
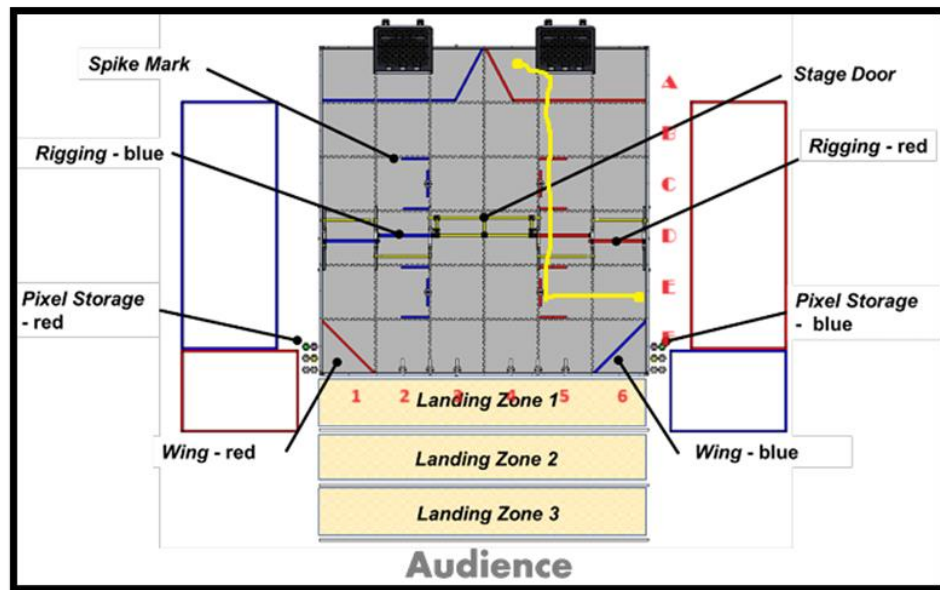

Figure 4.3(a) Centre pixel

The image's central layout acts as a reference point. The position of the randomization object on the central spike is indicated by an April tag. The robot starts moving from tile E6 in the center. It advances for 3 seconds, putting a pixel on the central spike. The robot then reverses back 2 inches in E5, performs a right turn for 3 seconds, and goes to sleep. Following that, it proceeds straight toward tile A5 for 30 seconds since it must traverse three 24inch tiles while aligning with the background on the backstage red spike mark. Backstage, the robot utilizes another April tag point, lowering pixel points if it aligns with the thing on the spike.

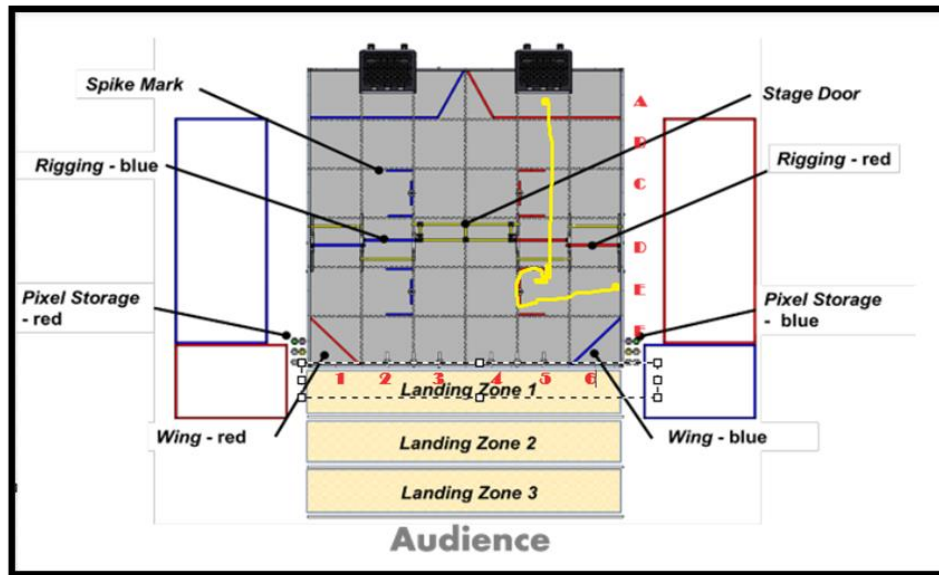b.   when the item is situated on the second tile's left side

Figure 4.3(b) Left side pixel

If the spike is placed at left side, then, the robot initiates its movement from tile E5 in the left. It travels forward for 3 seconds, placing a pixel on the left spike and makes a right turn at E5, the the robot then reverses back 2 inches in E5, executes a right turn for 3 seconds, with sleep time. Subsequently, it moves straight toward tile A5 for 30 seconds, aligning with the backdrop on the backstage red spike mark. The robot uses another April tag point backstage, dropping pixel points if it centers with the object on the spike. Finally, the robot parks at A4 followed by a left turn towards A4 and moving forward of 3 inches and a right turn on A4.

c. When the item is situated on the field's right side of the second tile.
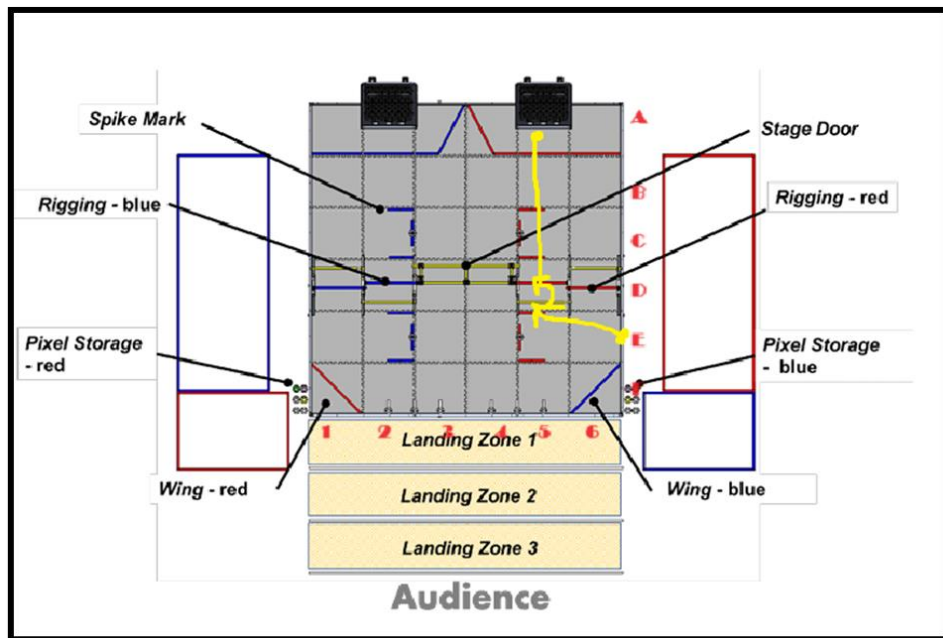

Figure 4.3(c) Right Side Pixel

Here the spike is placed at right side then, the robot initiates its movement from tile E5 in the right. It travels forward for 3 seconds, placing a pixel on the left spike, then it moves straight toward tile A5 for 30 seconds, aligning with the backdrop on the backstage red spike mark. The robot uses another April tag point backstage, dropping pixel points if it centers with the object on the spike. Finally, the robot parks at A4 followed by a left turn towards A4 and moving forward of 3 inches and a right turn on A4.

Autonomous FTC Code:

Op Mode Declaration and Initialization
@Autonomous (name="Robot: Autonomous_Opcode", group="Team2")
public class Autonomous_Opcode extends LinearOpMode

The @Autonomous annotation indicates that this class represents the robot's autonomous Op Mode. The name parameter defines the name of the Op Mode, which is "Robot: Autonomous_Opcode", and the group parameter sets the Op Mode in the "Team2" group.
Declaration of Class: public class Autonomous_Opcode enlarges LinearOpMode specifies an Autonomous_Opcode class that extends LinearOpMode. This signifies that the class inherits functionality for a linear Op Mode, which often entails executing a series of activities sequentially.

Configuration of the Motor and Encoder
private DcMotor leftDrive = null;
private DcMotor rightDrive = null;
private ElapsedTime runtime = new ElapsedTime();
static final double COUNTS_PER_MOTOR_REV = 1440;   // eg: TETRIX Motor Encoder
static final double DRIVE_GEAR_REDUCTION = 1.0;   // No External Gearing.
static final double WHEEL_DIAMETER_INCHES = 4.0;   // For figuring circumference
static final double COUNTS_PER_INCH = (COUNTS_PER_MOTOR_REV *
DRIVE_GEAR_REDUCTION) /
*   (WHEEL_DIAMETER_INCHES * 3.1415);*
static final double DRIVE_SPEED = 0.6;
static final double TURN_SPEED = 0.5;

leftDrive and rightDrive are DcMotor class instances that represent the left and right motors.
runtime is an instance of ElapsedTime that can be used in code to measure elapsed time.
COUNTS_PER_MOTOR_REV is the number of encoders counted per motor shaft rotation.
The gear ratio between the motor shaft and the wheel is represented by DRIVE_GEAR_REDUCTION.
WHEEL_DIAMETER_INCHES is the wheel's diameter in inches.
COUNTS_PER_INCH is computed by dividing the total encoder counts per revolution by the circumference of the wheel.
DRIVE_SPEED and TURN_SPEED are constants that reflect the speed of the robot while driving and turning.

Motor Initialization in runOpMode Method
public void runOpMode()
{
*   // ... (previous declarations)*
* leftDrive = hardwareMap.get(DcMotor.class, "left_drive");*
* rightDrive = hardwareMap.get(DcMotor.class, "right_drive");*
* leftDrive.setDirection(DcMotor.Direction.REVERSE);*
* rightDrive.setDirection(DcMotor.Direction.FORWARD);*

```
    leftDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    rightDrive.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    leftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    rightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
  }
```

The code begins the runOpMode method by collecting instances of the left and right motors via hardwareMap.get(DcMotor.class, "left_drive") and hardwareMap.get(DcMotor.class, "right_drive"). In particular, the left motor's direction is adjusted to REVERSE to match its actual orientation. Importantly, the encoder modes for both motors are then reset and set to RUN_USING_ENCODER, allowing for accurate motor control via encoder feedback during autonomous operations. This simplified approach ensures that the motors are expertly adjusted, ready for precise and controlled movements in autonomous mode.

OpenCV Camera and AprilTag Constants:

OpenCvCamera camera;
AprilTagDetectionPipeline aprilTagDetectionPipeline;
double fx = 822.317; double fy = 822.317; double cx = 319.495; double cy = 242.502;
double tagsize = 0.166;
int left = 1;
int middle = 2;
int right = 3;

In this section, we define variables for the OpenCV camera and declare constants that are required for camera calibration and AprilTag detection. To accurately describe the camera's behavior, the intrinsics of the camera, such as focal lengths (fx, fy) and main points (cx, cy), are defined. Furthermore, the 'tagsize' constant determines the size of AprilTags, which is required for exact detection. Additionally, unique IDs are assigned to the left, middle, and right tags, allowing for proper tracking inside the camera feed. These variables and constants, when combined, provide the backbone of a successful computer vision system, ensuring correct calibration and reliable AprilTag detection and tracking in a variety of locations.

AprilTag Detection and Robot Movement

AprilTagDetection tagOfInterest = null;
boolean tagFound = false;
boolean aligned = false;

These variables hold information on the detected AprilTag, whether a tag is located, and whether the robot is aligned with a tag.

runOpMode Method

@Override
public void runOpMode()
public void tagInformation()
void tagToTelemetry(AprilTagDetection detection)
public void goTowardsTag(int tag)
public void dropPixel()
public void encoderDrive(double speed, double leftInches, double rightInches, double timeoutS)

The main method that is called robot initialization, manages the complete robot system. It includes the logic for hardware initialization, OpenCV camera setup, AprilTag detection, and subsequent tag-based actions.

tagInformation( ):
Description: Takes care of AprilTag detections.
Functionality: Handles AprilTag information interpretation, allowing subsequent decision-making based on detected tags.

tagToTelemetry( ):
Sends information about the AprilTag to telemetry.
Functionality: Sends AprilTag data to telemetry system for real-time monitoring and analysis.

goTowardsTag( ):
Description: Contains instructions for movement based on the detected tag.
Functionality: Defines precise robot movements and actions associated with the recognized AprilTag, allowing for dynamic responses to the environment.

dropPixel( ):
It is used to control a servo.
Functionality: Controls the servo mechanism, giving fine control over object dropping or manipulation based on the robot's purpose.

encoderDrive( ):
Describes how motor encoders are used to control the robot's movement.
Functionality: Uses motor encoder feedback to control the robot's motions precisely, ensuring accurate and dependable navigation inside the surroundings.

Camera Initialization and Tag Detection:

Int cameraMonitorViewId=
hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId","id",
 hardwareMap.appContext.getPackageName());
camera = OpenCvCameraFactory.getInstance().createWebcam(hardwareMap.get(WebcamName.class,
"Team2_webcam"), cameraMonitorViewId);
aprilTagDetectionPipeline = new AprilTagDetectionPipeline(tagsize, fx, fy, cx, cy);
 camera.setPipeline(aprilTagDetectionPipeline)
camera.openCameraDeviceAsync(new OpenCvCamera.AsyncCameraOpenListener()
{
 @Override
 public void onOpened()
 {
 camera.startStreaming(720,500, OpenCvCameraRotation.UPRIGHT);
 }
 @Override
 public void onError(int errorCode) { }
});
telemetry.setMsTransmissionInterval(50);

To identify the camera monitor view in the user interface, the system dynamically obtains the 'cameraMonitorViewId'. It then uses the 'createWebcam' function to make an OpenCVcamera instance named "Team2_webcam." An 'aprilTagDetectionPipeline' is initialized with tag detection settings. The'setPipeline' function connects the AprilTag recognition pipeline to the camera, and the 'openCameraDeviceAsync' function simultaneously opens the camera device, starts streaming in Driver Hub upon successful opening.

INIT-loop and Tag Detection

while (!isStarted() && !isStopRequested())

{

   tagInformation();

   if(tagFound)

   {

     telemetry.addLine(String.format("\nApril_Tag_Detected is =%d", tagOfInterest.id));

     tagToTelemetry(tagOfInterest);

     // Function to turn continuously until x-axis lies between -0.6 to 0

   }

   telemetry.update(); sleep(20);}

During the initialization phase, this loop runs until the OpMode is started or terminated (!isStarted() &&!isStopRequested()). TagInformation is called within the loop to detect and analyze AprilTags. If a tag is discovered (tagFound), telemetry is updated with the observed tag's information (tagToTelemetry). A comment implies that there should be a function that turns the wheel indefinitely until a certain condition is met.

Execution after OpMode Start

if(tagOfInterest.id == left)

   {

     telemetry.addLine("Detected_Left April Tag");

   telemetry.update();

     goTowardsTag(left);

   }

   else if (tagOfInterest.id == middle)

   {

      telemetry.addLine("Detected_Middle April Tag");

   telemetry.update();

      goTowardsTag(middle);

   }

   else if (tagOfInterest.id == right){

      telemetry.addLine("Detected_Right April Tag");

   telemetry.update();

     goTowardsTag(right);}}

public void tagInformation(){

   ArrayList<AprilTagDetection> currentDetections = aprilTagDetectionPipeline.getLatestDetections();

   if(currentDetections.size() != 0) {

    for(AprilTagDetection tag : currentDetections)

    { if(tag.id == left || tag.id == middle || tag.id == right){

      tagOfInterest = tag;

      tagFound = true; break;}}}}

Within the autonomous robot control code, the 'tagInformation()' method is a critical component in the AprilTag detecting process. The procedure reads the most recent AprilTag detections from the 'aprilTagDetectionPipeline' and checks the ID of each detected tag. It particularly looks for AprilTags

with predetermined IDs, such as 'left', 'center', or 'right'. If a tag matching one of these IDs is discovered, the method marks it as the 'tagOfInterest' and sets the 'tagFound' flag to 'true'. Importantly, the technique exits the loop once the first relevant AprilTag is identified, assuming that only one tag of interest occurs in a given frame. This capability is frequently used during the initialization loop, checking for particular AprilTags continually prior to the official start of the OpMode.

Method: tagToTelemetry(AprilTagDetection detection)

void tagToTelemetry(AprilTagDetection detection)

{ telemetry.addLine(String.format("\nDetected tag ID=%d", detection.id));

telemetry.addLine(String.format("Translation X: %.2f feet", detection.pose.x * FEET_PER_METER));

telemetry.addLine(String.format("Translation Y: %.2f feet", detection.pose.y * FEET_PER_METER));

telemetry.addLine(String.format("Translation Z: %.2f feet", detection.pose.z * FEET_PER_METER));

}

This method acknowledges an AprilTagDetection object as a parameter and updates telemetry with information about the detected AprilTag. telemetry.addLine(String.format("Detected tag ID=%d", detection.id)): Adds a telemetry line displaying the ID of the detected AprilTag.

telemetry.addLine(String.format("Translation Y:%.2f feet", detection.pose.y * FEET_PER_METER)): Adds a telemetry line that displays the detected tag's Y-axis translation in feet.

telemetry.addLine(String.format("Translation Z:%.2f feet", detection.pose.z * FEET_PER_METER)): Inserts a telemetry line reporting the detected tag's Z-axis translation in feet.

This method is in charge of delivering real-time feedback on the ID and translation of the detected AprilTag in the X, Y, and Z directions.

Method: goTowardsTag(int tag)

public void goTowardsTag(int tag){

if(tag == left){

. (code for the left tag)

} else if (tag == middle) {

(code for the middle tag)

} else if (tag == right) {

(code for the right tag) }}

This method is intended to do specified actions based on the ID of the detected AprilTag. There are three scenarios considered: left, middle, and right, with different sets of moves established for each.

Scenario tag == left: The robot is told to go forward (encoderDrive) and turn based on specified values.

It contains calls to the dropPixel function as well as other movements.

Scenario tag == middle: The robot is ordered to move forward, turn, drop a pixel, and conduct other moves, similar to the left scenario.

Method: dropPixel()

public void dropPixel(){

if (rampUp) {

```
    position -= INCREMENT ;
    if (position <= MIN_POS ) {
      position = MIN_POS;
      rampUp = !rampUp;  } }
  servo.setPosition(position);
  sleep(CYCLE_MS);
  idle(); }
```

The 'dropPixel()' method gradually moves a servo to simulate dropping an item, often known as a "pixel." During the ramp-down phase, it drops the position by a predetermined increment ('INCREMENT') or sets it to a minimal threshold ('MIN_POS') if it falls below a certain threshold. The approach manages the 'rampUp' condition efficiently, assessing if the servo is still in the descent phase. Following the setting of the servo position, there is a brief pause ('CYCLE_MS'), followed by idling to accommodate any asynchronous processes. In essence, this strategy assures that the servo descends in a controlled and steady manner, simulating the deliberate release of an object.

Method: encoderDrive(double speed, double leftInches, double rightInches, double timeoutS)

```
public void encoderDrive(double speed, double leftInches, double rightInches, double timeoutS)
{
  int newLeftTarget;
  int newRightTarget;
    if (opModeIsActive())
  {
    newLeftTarget = leftDrive.getCurrentPosition() + (int)(leftInches * COUNTS_PER_INCH);
    newRightTarget = rightDrive.getCurrentPosition() + (int)(rightInches * COUNTS_PER_INCH);
    leftDrive.setTargetPosition(newLeftTarget);
    rightDrive.setTargetPosition(newRightTarget);
    leftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    rightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        runtime.reset();
    leftDrive.setPower(Math.abs(speed));
    rightDrive.setPower(Math.abs(speed));
    while (opModeIsActive() &&
        (runtime.seconds() < timeoutS) &&
        (leftDrive.isBusy() && rightDrive.isBusy())) {
      telemetry.addLine("Robot is in motion");
      telemetry.update();
    }
```
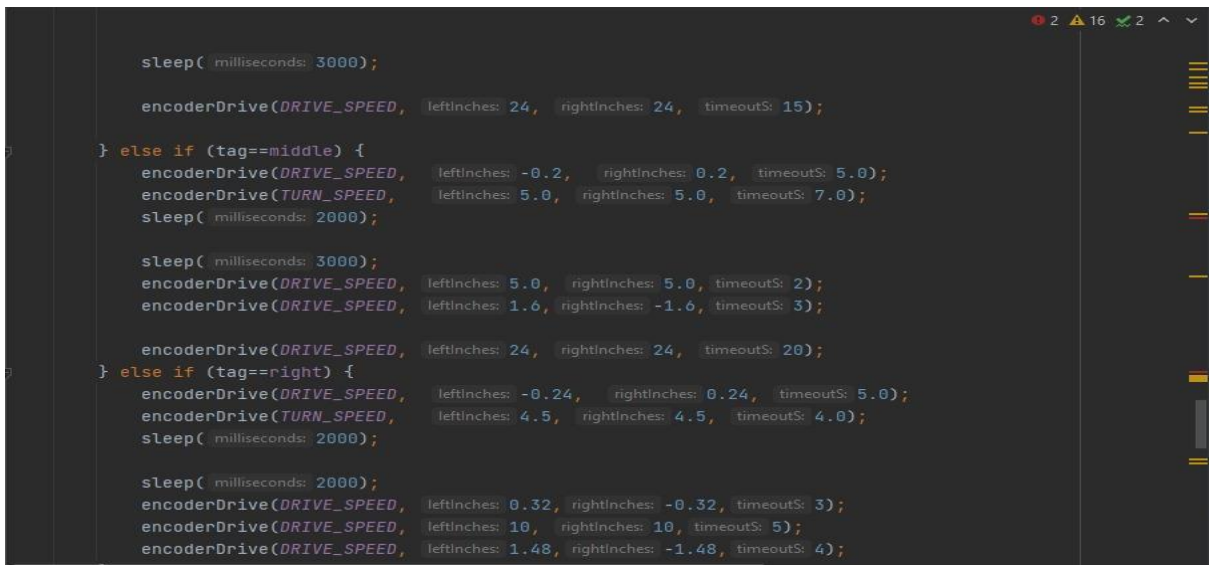
```
        leftDrive.setPower(0);
      rightDrive.setPower(0);
      leftDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
      rightDrive.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
   sleep(250);
    }}
```

The 'encoderDrive()' method allows for controlled robot movement to precise positions by using encoder counts. Based on the intended distances, it determines new target positions for the left and right motors. The motors are set to move to these points, and motion starts at the preset speed. Within a time limit, the approach observes motion progress and updates telemetry accordingly. When the motion is finished, the motors stop and their run modes revert to encoders. A slight interval allows the motor to settle before beginning. This method includes robot movement logic that is accurate and controlled.

Figure 4.4 Distance with encoder coder middle and right

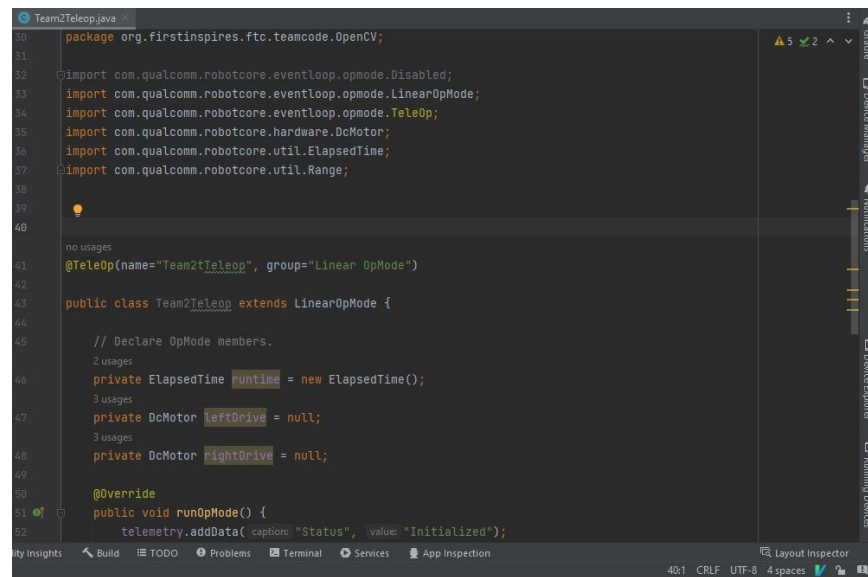

Figure 4.5 April Tag detecting code
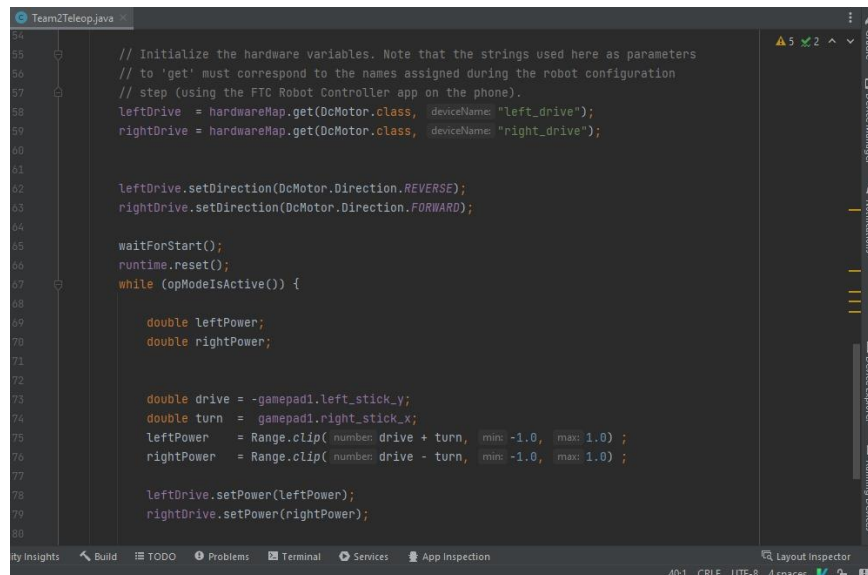
Teleop:

Figure 4.6 Teleop

Figure 4.7 Teleop Mode

This code is for controlling a robot with a game controller. The robot has two motors, one on each side, one on each side. The program waits for a signal to begin before constantly checking the gaming controller. The left joystick controls the robot's forward and backward movement, while the right joystick controls its turning. Based on the joystick inputs, the programming adjusts the power to the motors. It also saves time and displays information on the driver's screen. To put it simply, it allows someone to drive the robot using a game controller during a tournament. The functions of each button on the remote control were indicated on the label. The left and right sticks in the code are set to y and x. Additionally, the gamepad's hand position was programmed to respond when the designated buttons were pressed. A

designated team member will control this part of the game. The robot has been given coding that allows it to be controlled and driven. We adjust speed to achieve the optimal movement to score points.

End Game

In the last stage of the robot game, the robot performs precise moves based on the location of a special object. If the object is in the center, the robot begins in the center, places a pixel on the middle spike, and then proceeds to a set location, aligning with a red mark to earn extra points. If the object is on the left side, the robot begins on the left, places a pixel on the left spike, and then repeats the process for points. If the object is on the right, the robot will begin on the right and perform identical behaviors.

These motions are meticulously designed to gain the greatest points by perfectly interacting with the set object. The usage of special markings known as April tags assists the robot in making these precise maneuvers in the game.

5   Internal/external Interface Impacts and Specification

In the field of robotics, teleinformatics is the process of gathering information from sensors and other sources on a robot and sending it to a distant location for the purposes of control, analysis, or monitoring. For real-time understanding of a robot's status, surroundings, and performance, teleinformatics systems are essential. The status of the motor, battery levels, sensor readings (such as temperature, LiDAR, and cameras), location, and other details can all be included in this data.

Through teleinformatics, operators or controllers can remotely check on the health of the robot, identify problems, make deft decisions, and even modify its behavior or mission parameters. It's crucial in situations like space exploration, deep-sea exploration, hazardous environments, and autonomous operations where direct human supervision or intervention may be difficult.

Depending on the range and particular needs of the application, wireless communication technologies like Wi-Fi, Bluetooth, cellular networks, or specialized communication protocols are frequently used to send the teleinformatics data transmitted from the robot. This information can be processed by other systems to enable automated decision-making, or it can be presented to human operators via control panels or user interfaces.

6   Design Units Impacts

Touch Sensor: A touch sensor in robots is a tactile input device that detects physical contact with its surface. When pressed or touched, it sends a signal to the robot's control system, enabling it to respond to touch-based interactions. Touch sensors are commonly used in robotics for tasks like collision detection, interaction with the environment, or implementing touch-sensitive controls in various applications.

Figure 6.1 Touch Sensor

Battery: The REV-31-1302 battery, often used in the context of the FIRST Tech Challenge (FTC), is a rechargeable lithium-ion battery pack designed for use with FTC robots. It provides the necessary power to drive the robot's motors and components during competitions. The specific details, specifications, and usage guidelines for the REV-31-1302 battery may be available in the official REV Robotics documentation or the FIRST Tech Challenge game manual. For the most accurate and up-to-date information, it is recommended to refer to REV Robotics' official resources or contact their support directly.

Figure 6.2 REV Robotics (REV-31-1302)

Motors: The motor operates at a voltage of 12 volts, making it suitable for robotics applications where this voltage is commonly used. The motor has a hexagonal output shaft, which is a common feature in robotics motors. The hex shape provides a secure connection and prevents slippage when attached to components like wheels or gears. The Core Hex motor is designed to be compatible with other REV Robotics components, allowing for easy integration into a robotics system. The motor is used to drive various mechanical components of a robot, providing the necessary power for movement and functionality. It is essential for tasks such as driving the robot, operating arms or mechanisms, and other motion-related functions.

Figure 6.3 REV Robotics Core Hex 12V DC Motor

Web Camera: A webcam is a device that provides visual images of the surrounding environment. For use as part of FIRST Tech Challenge teams must use a COTS UVC (USB Video Class) Compatible Camera. This

device can be connected directly to the REV Control Hub or to the Robot Control system via a powered USB hub <RE14>. This device is intended to be used in vision related tasks.

Figure 6.4 Web camera

Acknowledgement

We Express our deep sense of gratitude to our learned guide **"Prof Xueqing Tang"** for your valuable help and guidance. We are thankful to you for the encouragement that you have given us in completing the project.
We are also thankful to all the other faculty and staff members of our department for their co-operation and help.
Lastly, we would like to express our deep apperception towards our teammates and our indebtedness to our parents for providing moral support and encouragement.

Thank you.

References

1. https://firstinspiresst01.blob.core.windows.net/first-in-show-ftc/game-manual-part-2-traditional.pdf

2. https://ftcdocs.firstinspires.org/en/latest/control_hard_compon/ds_components/components/components.html

3. https://docs.revrobotics.com/duo-control/control-hub-gs/driver-station-pairing-to-control-hub

4. https://www.youtube.com/watch?v=6e-5Uo1dRic

5. https://git-scm.com/downloads

6. https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio#2

7. https://www.youtube.com/watch?v=GXeSsbGXURM

8. https://www.revrobotics.com/ftc/motion/motors-servos/

9. https://youtu.be/RGOj5yH7evk? si=m5bVb17mu-MXwflS

10. https://youtu.be/1WnGv-DPexc?si=oHTw5AgkhDYvnnvV

Appendices

Link to code Text is available in the GitHub Link below:

https://github.com/nazmashaik29/Team2/tree/main/FtcRobotController-9.0