Torrey Pines High School Botball

Software Design Document

2009 Botball Season Team Number 09-0207

1. Introduction

The following document is the Software Design Document for the Torrey Pines High School Botball Team. It includes the different methods to be implemented by the team and pseudocode of the game strategy via flowcharts.

1.1 Purpose

The purpose of this document is to plan out the different methods to be implemented by us, The Torrey Pines High School Botball Team. It is also meant to demonstrate our plan of action to KIPR for software documentation so they can have a better understanding of how our methods contribute to our game strategy.

1.2 What is Botball?

Botball is a robotics competition in which high school and middle school students take part. Students are given a kit which they must build their robots from. No extra parts are allowed in the competition. Students are given an IRobot Create and other motors that can be used to make another robot. Students must also document their documents and procedures throughout their building and programming process. The robot must be autonomous, meaning that the robot must think for itself and students are not able to move the robot via remote control. This document illustrates how Torrey Pines High School Botball will attempt to implement their autonomous robots.

1.3 Definitions, Acronyms and Abbreviations

Tribbles – Small fuzzy balls that are either green or orange. The tribbles are worth a small amount of points in this year's Botball game. Green Tribbles represent clean energy, while orange poms represent fossil fuels.

Waters – Blue sponge balls that are much larger than tribbles. They are used to represent water and are worth much more than tribbles are. However, they are also harder to collect.

Turbine – Pencils with a pinwheel attached to the eraser. These worth as much as waters but are the hardest items to collect in this year's game.

Legobot – One of Torrey Pines High School's two robots. The robot driving base and arms are made of mainly vex pieces with a few Lego parts. The robot's goal during the seeding round is to push the waters into the path of the Create. The robot's goal during the double elimination round is to go to the opposing slope and block them from dropping of resources to score points.

Create – The Create is the second of Torrey Pines High School's robots. The driving base is an IRobot Create and has servos on it that help attach a detachable bulldozer. The robot's goal during the seeding round is to get as many points as possible with a combination of tribbles, waters, and turbines. During the double elimination round, the robot will collect 5 green tribbles, 5 orange tribbles, possibly a water or two, and one turbine.

2. References

KISS-C Programmers Manual for the CBC Botball Controller – a programming manual provided by KISS institute to give students a reference to the different methods supplied and different programming techniques in the C programming language.

Botball 2009 Game File – The game file explains all the rules of the game and supplies the different game pieces and their value.

3. Legobot

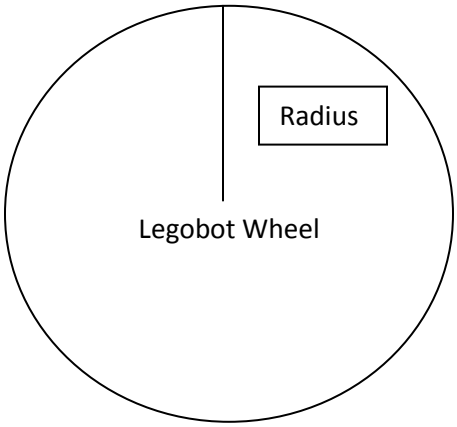
3.1 Legobot Abstract

Our Legobot will be designed and programmed to go to the enemy slope and block the opponent. Our robot has a very simple job and therefore, our robot is task oriented. Driving to the opposing slope and deploying its blocking arms are the only two tasks that our robot has.

3.2 Methods

void moveDistance(float distance)

To drive, we will implement several extra methods that will help the Legobot drive to the slope faster. The first method we will implement to move the Legobot is named moveDistance. To implement this method we will use trigonometry. We measured the diameter of the wheel and used it to find circumference. Using the circumference, we could then find the distance in millimeters that each wheel travels in one revolution or in this case, every 1100 ticks. Using this conversion factor, we can pass in the distance as a float and multiply it by the amount of ticks per millimeter. We can call the library method move to position using the amount of ticks to be traveled. It is essential to be able move the Legobot straight.



Legobot Wheel radius = 35 mm

Legobot Wheel Circumference = 219.911 mm

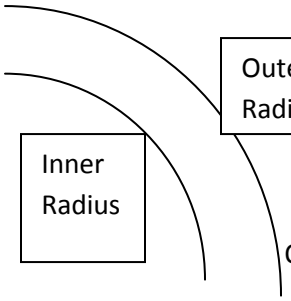
1 circular revolution = 1100 ticks

Moving 1100 Ticks = Moving 219.911 mm

Conversion Factor = 5.002 ticks/mm

void turnArc(boolean leftArc, float outerRadius, int speed, float moveDistance)

Another method is an arcing method named turnArc. The method will take in a boolean which will determine if we are arcing left or right, the radius of the outer wheel, the velocity of the outer wheel, and either the distance to travel of the outer wheel or the number of degrees of the circle the robot will cover. We will use the information we had in the distance method to move a certain distance on each wheel. We will use a ratio between the different radii of the arc and then use the ratio to calculate the speed and distance of both of the wheels. To calculate the ratio, we will input the outer radius. We will take the outer radius, subtract the distance from the center of one wheel to the other and divide the difference by the outer radius. This ratio allows us to calculate the velocity of the inside wheel and the distance needed to travel. This function will allow us to move to the slope by moving in a fluid motion, rather driving straight, turning, then driving straight again.



Outer Radius = R Legobot Diameter = 120

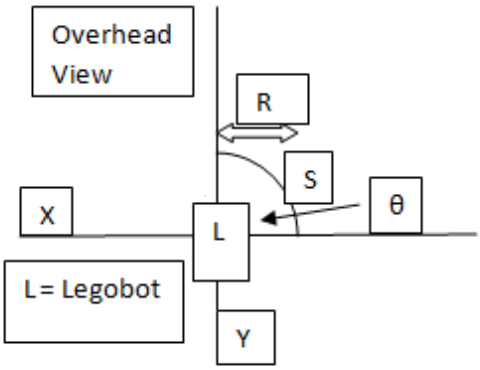
Inner Radius = r = R – 120 Ratio = r/R

Outer Travel Distance = TD Inner Travel Distance = (TD * ratio)

Outer Wheel Velocity = OV Inner Wheel Velocity = (OV * ratio)

void turn(float degrees)

In addition to the arcing method, we will implement a turning method that will turn the robot in place. In order to implement the turn function, we will use the ticks per millimeter conversion factor we calculated when implementing the moveDistance method. To turn a certain amount of degrees in place, we will call the function with a certain amount of degrees. The function will then find arc length of the angle, which would also be the distance traveled by one wheel. We will define the constant ticks per degree as a conversion factor. We can turn left by calling the function with positive degrees and right by calling the function with negative degrees with respect to the unit circle. We will use the conversion factor and multiply it with a certain number of degrees. The function will then use the kiss-c library method move relative position to move both the motors a certain amount of ticks in opposite directions. This function will be useful when driving up to the base of their slope and turning 90 degrees in place.



$R\theta = S$ R = robot radius = 120mm θ = Degrees to turn

S = arc length = distance travelled by one wheel

Distance travelled by one wheel = T / (TPM)

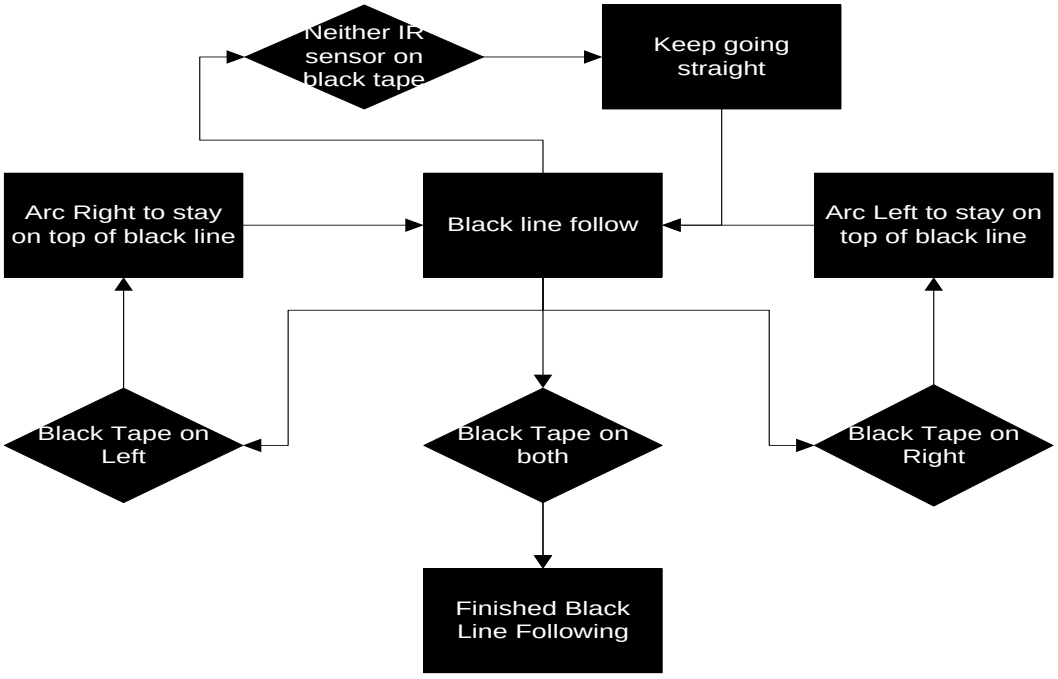
T = ticks TPM = ticks per millimeter (distance method)

$\therefore T/\theta = TPD$

Conversion Factor = 5.4817 ticks/degree

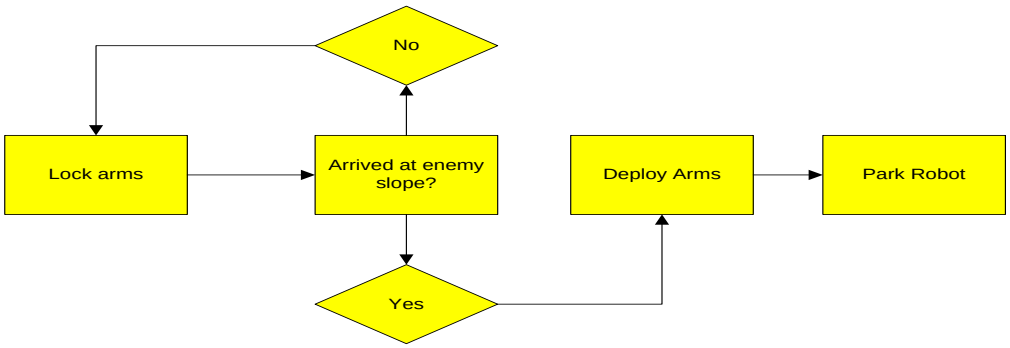
void followBlackLine(void)

The Legobot will use a line following algorithm to go to the base of the opposing team's slope during the double elimination rounds. To follow the black line leading up to the slope, we will use two IR top hat sensors on the left and right sides of the robot. The values for the IR sensors are usually under 150, but when the sensor goes over a black line, the values shoot up past 600. If the IR sensor on the left side senses a black line, the robot will start arcing right in order to stay on path. It arcs in the opposite direction when the right IR sensor detects a black line. The robot will keep correcting its path until it senses black on both sides. In other words, the only time the robot senses black lines on both sides is at the dividing area at the base of the slope.



void lockArms(void) and void releaseArms(void)

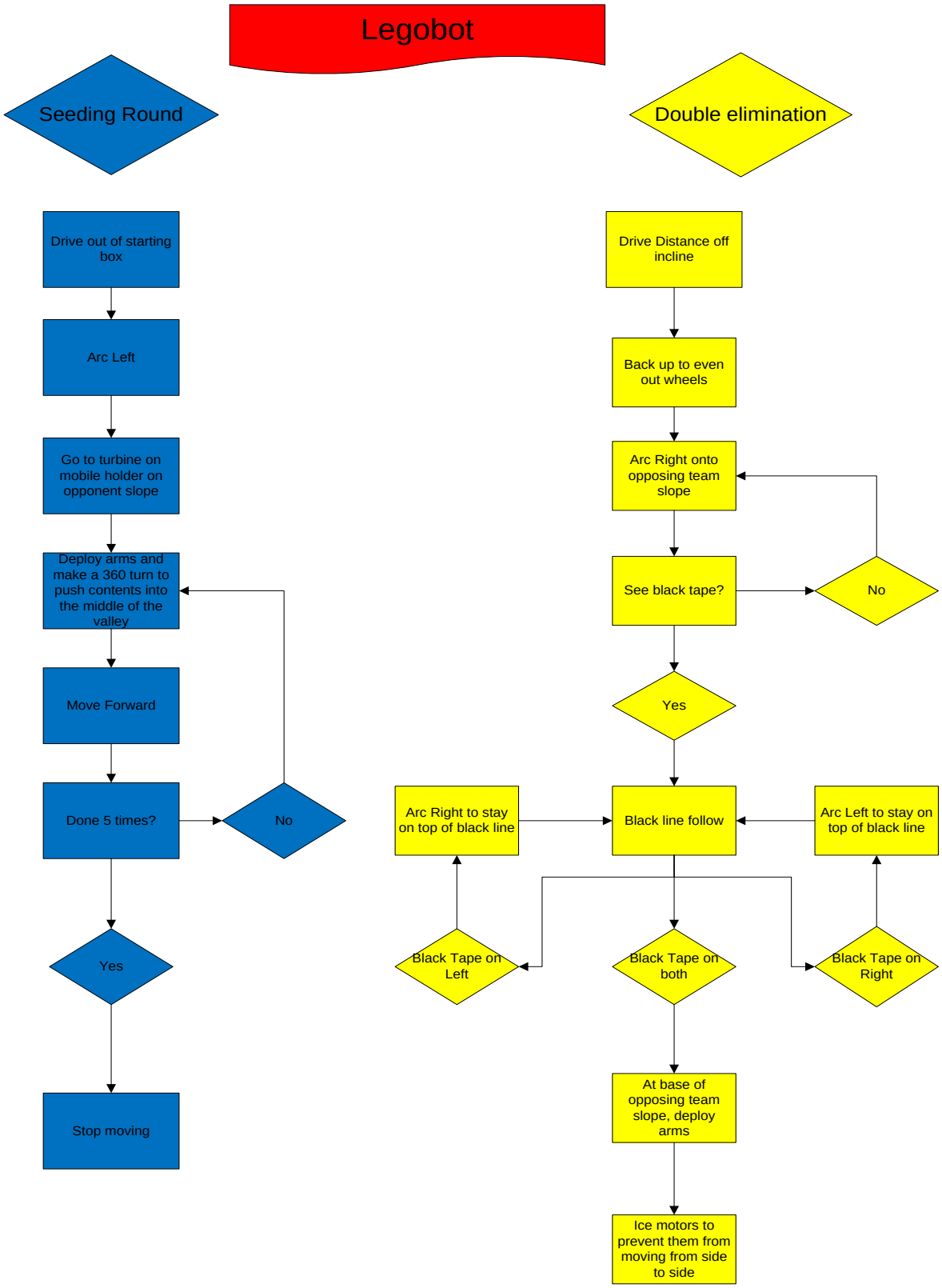
Initially, the arms of the robot are held vertically using pins which are attached to servos. There will be two positions defined for both servo motors. The first position to be defined to be the point where the pin keeps the arm locked, and hanging vertically. The second servo position is defined to be the point where the pins are pulled back and the arms are dropped. The servos will keep the arms at the locking position while the robot is travels to the base of their slope. When the robot is ready to deploy its arms, it will move the servos to detach the pins.



3.3 Strategy

The robot's goal during the seeding round will be to push the waters into the path of the Create. The robot's goal during the double elimination round will be to go to the opposing slope and block them from dropping of resources to score points.

3.4 Strategy Walkthrough



4. Create

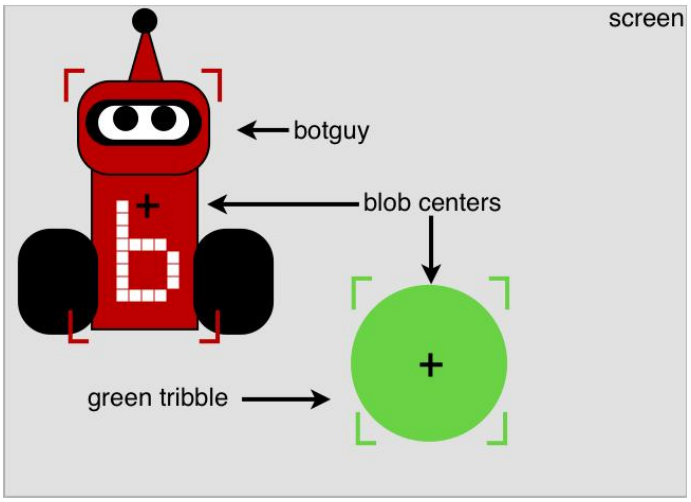
4.1 Create Abstract

The Create will be based on a mostly hard-coding, task-oriented architecture. The sensors on the robot will use include a camera as well as the built-in bump sensors. The Create will push objects onto the peak and subsequently block the starting box.

4.2 Methods

int centerObject(int channel)

This method will make the Create to spin until it is centered on an object of specified color. This method will work by identifying the largest blob of the specified color on the screen. It will then identify the center x-coordinate of that blob. The method will compare it to the center x-coordinate of the screen, and depending on whether the center of the blob is lesser or greater than the center of the screen, the Create will spin in a particular direction until the two center of the blob is the same value.

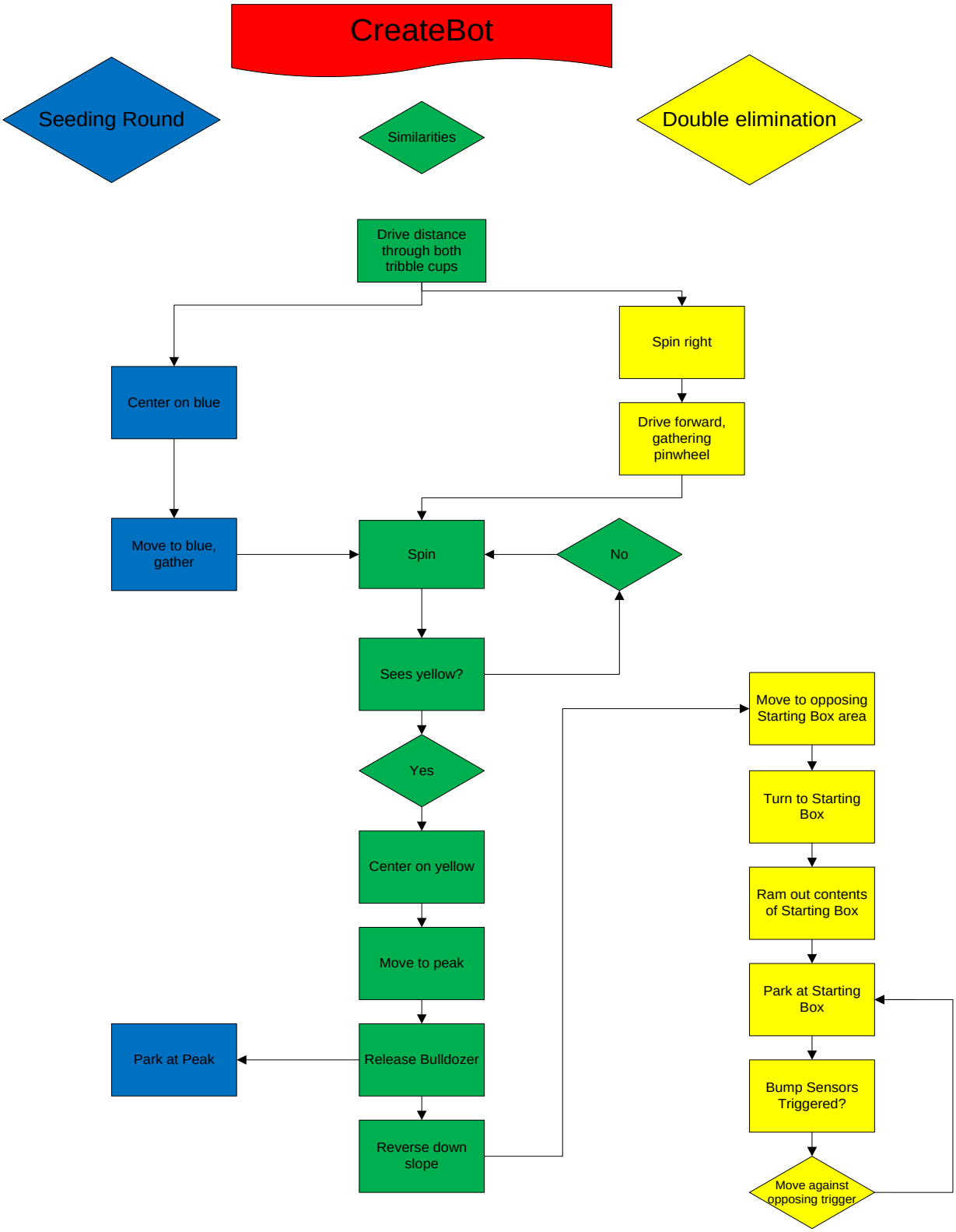


For turnArc method, see Legobot.

4.3 Strategy

The Create's strategy will be to generally score points in the peak area. During the seeding round, because of less time constraint due to the lack of an opposing team, the Create will attempt to score all tribble cups, water, as well as Botguy. However, during the double elimination rounds, the Create will attempt to score only 2 tribble cups and possibly a wind turbine, but it will also block the opposing side's starting box.

4.4 Strategy Walkthrough

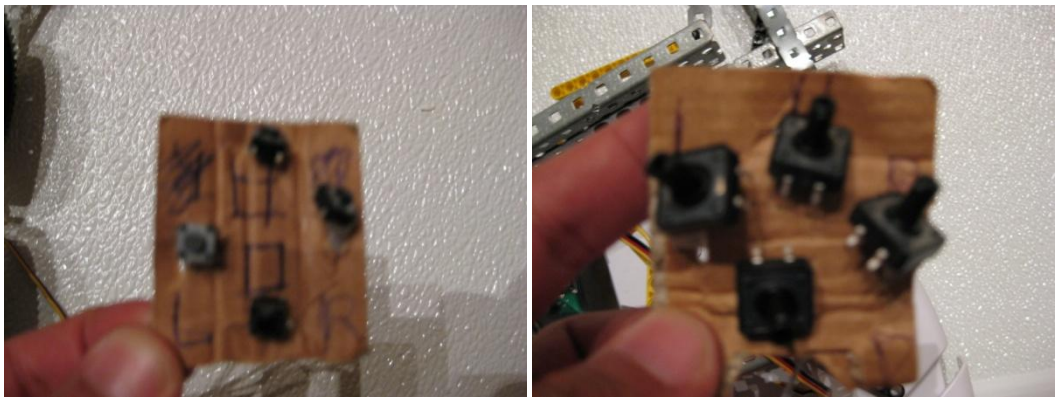


5. D-pad Robot Controller

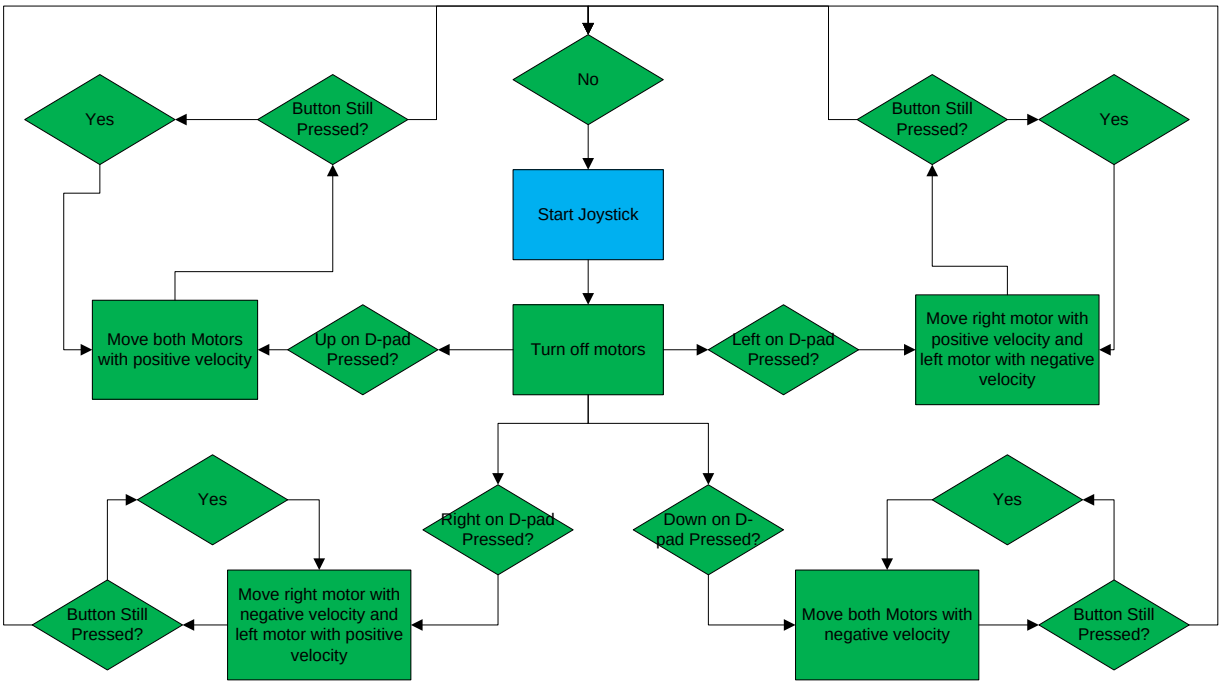
5.1 D-pad Robot Controller Abstract

The D-pad Robot Controller is a d-pad that the Torrey Pines High School Botball Team has constructed in order to controller their robots remotely. It is made up for four touch sensors that are mounted on square piece of cardboard. The D-pad helps the team test the limits of their robots. It also helps them write pseudocode for their robots and find the fastest paths for the proposed testing strategies.

5.2 D-pad Samples



5.3 D-pad Simulation



6. Implementing and Conclusion

The Torrey Pines High School Botball Team will implement the functions stated above. These functions will be the foundation of both the robots' behaviors. Most of the Legobot's methods are based on moving the robot from point A to point B. The moveDistance method will used when moving the Legobot in a straight line. The turn method will work in conjunction with the moveDistance method to turn the robot in place. Instead of using these methods together, it is possible to using turnArc instead. The method turnArc allows the robot to arc, meaning that it can move a distance while turning. The arcing method will also be helpful when writing the black line following. The arcing allows the Legobot to make minimum turn changes and also allows it to continually move forward instead of stopping and turning. Our methods to lock and release the arms are very simple and important. The servos on each side of the Legobot help lock the arms in place by setting the servo to a certain position. When they need to be released, the servo positions are changed and which allows the arms to drop via gravity. The driving methods require knowledge of trigonometric concepts and other mathematical applications.

The Create has very straightforward moving and turning methods, and its' task-oriented nature makes its movement methods simple to code. The move method will be used to move the robot in a straight line and the turn method will be used to allow the Create to spin. The center method uses the turn method in order to spin until the desired object is centered. This will help in centering the water, the Botguy, and the yellow foundations at the peak.

Both robots make use of a makeshift directional pad we have created. Using four bump sensors on each d-pad, we allow the choosing of several programs, including certain servo tests and motor tests, as well as a mode where we have direct control over the robot's movement (moving forward, backward, and turning left or right).

The combination of our proposed methods, proposed strategy, and testing devices make us confident that we can effectively implement our robots. With testing, we can perfect the execution of our proposed strategies. We hope that our strategies are pulled off in stylish faction at the Southern California Botball Regional's.