MOE, Miracles of Engineering

FTC Team 365

2018-19 Control Award Submission

Introduction

Throughout the design of our robot, we have kept one universal theme in mind.

User Friendliness: The measure of how robust, simple, easy to maintain, and easy to use a robot is.

To accomplish this goal of user friendliness in Autonomous and TeleOp, we have tried to keep the number of important components on the robot (sensors, motors, etc...) to a minimum while still vying to accomplishing our goals in mind. The result has been a robot that places more importance on intricate algorithms than sensors.

Our robot does utilize a good number of sensors, but wherever one can be omitted (for example: a camera rather than a color sensor), we take that option. This results in less environmental variables that can impact robot performance, as the robot relies on its algorithms and math to do computation in the place of sensors that could sometimes provide faulty data.

When driving the robot, we try to keep controls as simple as possible to allow the driver to focus on making important decisions rather than be distracted or bothered with the controlling of the robot.

Along with programming for the sake of the robot in competition, we have also programmed for the sake of learning (such as creating our own Neural Network!) to involve ourselves in other forms and kinds of programming. For an explanation on our thought process & more experimental procedures, view the Additional Summary Information. Also, below most titles will be a listing of notebook pages grouped together by what stage in the development process they show.

The 6 main sections are as follows:

1. Autonomous Objectives
2. Sensors Used
3. Key Algorithms
4. Driver Controlled Enhancements
5. Autonomous Program Diagrams
6. Additional Summary Information

Table of Contents

1.0 Autonomous Objectives

2.0 Sensors Used

3.0 Key Algorithms & Constructs

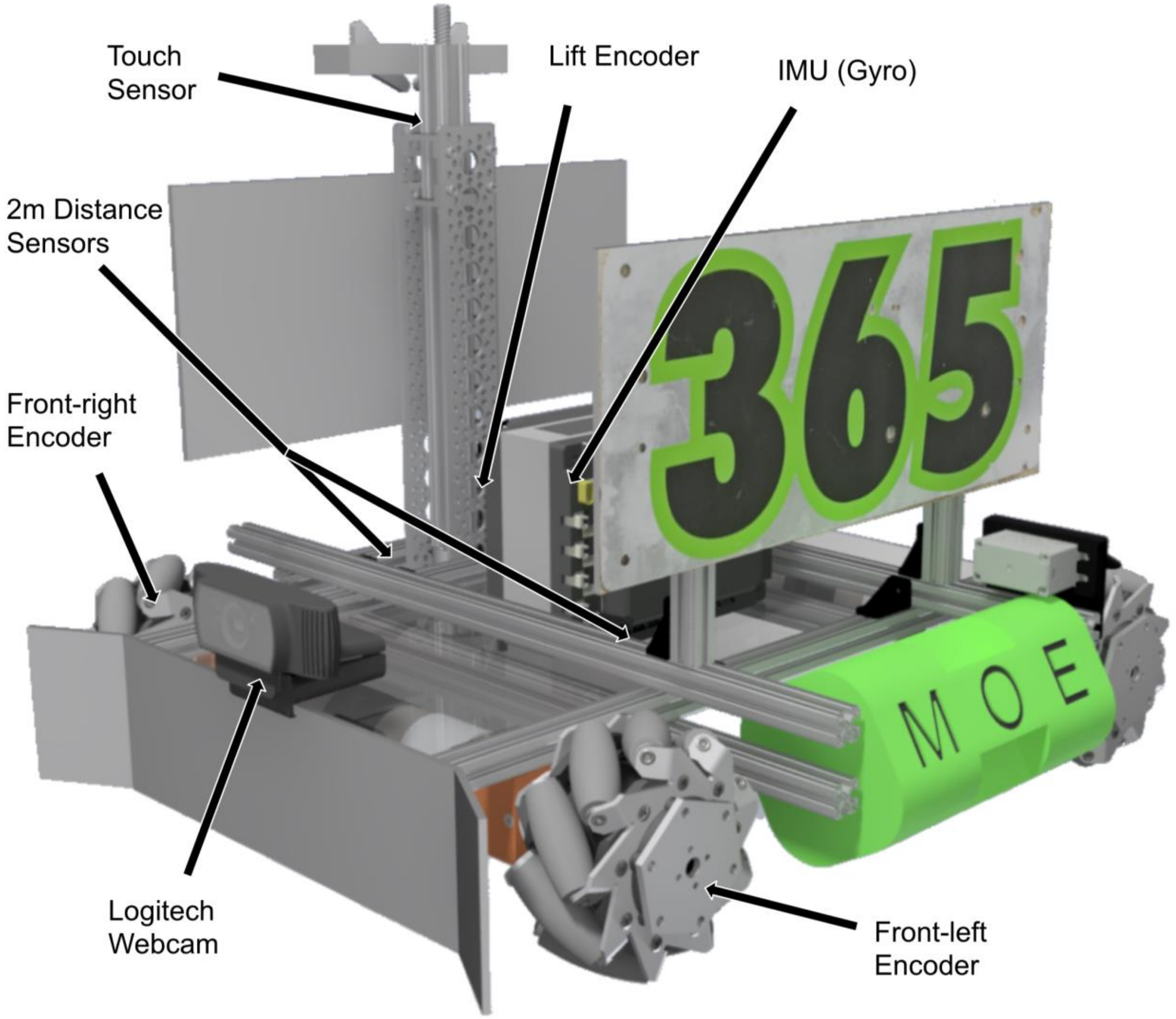
Autonomous Objectives

The following objectives are what we planned for in our robot's autonomous modes.

- **Autonomous Routine: (82 pts.)**
    - Landing – dropping off of the lander (30 pts.)
    - Sampling – knocking off the gold mineral (25 pts.)
    - Sampled gold mineral placed in depot (2 pts.)
    - Claiming – dropping the Team Marker in the depot (15 pts.)
    - Parking – ending autonomous in the crater (10 pts.)
- **Algorithmic & Programming Objectives:**
    - Establishing Field Grid & MOEPS (MOE Positioning System)
    - Localization
    - Multithreading
    - Accurate Turning Methods
    - Accurate Pathfinding with A* and Dijkstra's Algorithms
        - Pathfinding Error Correction
        - "Rotational Symmetry"
        - Realignment
    - Error Correction & Fallback Plan B Routines

Although not completely relevant in explaining the controls and actions of the robot, we included additional algorithms and details in our programming process that we felt to be of importance in the *Additional Summary Information*.

Sensors Used



Encoders (3)

2: Encoders placed on motors involved with the robot’s mecanum drive. One encoder is on the front-left wheel, while the other is on the front-right wheel.

1: Encoder placed on the lift motor involved with dropping/hanging. The encoder is used for precise, controlled motion of the lift motor.

Inertial Measurement Unit (IMU) (1)

1: IMU build into the REV Expansion Hub. This IMU is effectively a gyro sensor, with the capability to measure rotation on 3 axes. We primarily use the horizontal axis, or the one that measures rotation parallel to the ground for accuracy in any turns or rotational movement of the robot.

Logitech Webcam (1)

1: Logitech Webcam used in the front of the robot. Using this rather than a phone camera allows for the phone to be safely protected within the robot, making sure that nothing goes wrong during the match. The Logitech Webcam is used for recognizing the Vumarks.

REV 2m Distance Sensor (2)

1: REV 2m Distance Sensor placed on the front side of the robot, facing the forward direction. The sensor is primarily used for telling distance away from the walls of the field.

1: REV 2m Distance Sensor placed on the right side of the robot, facing the right direction. The sensor is primarily used for telling distance away from the walls of the field.

Touch Sensor (1)

1: Touch Sensor placed near the top of the Tetrix channel used for holding the linear actuator used in dropping/hanging. The sensor is used for safe and automated resetting of the lift in hanging. Since the lift has to have the same starting point across all robot autonomous modes, a standard starting point is important.

Key Algorithms & Constructs

Field Grid & MOEPS (MOE Positioning System)

Conceptualization and implementation: C7, C8, C13, C14

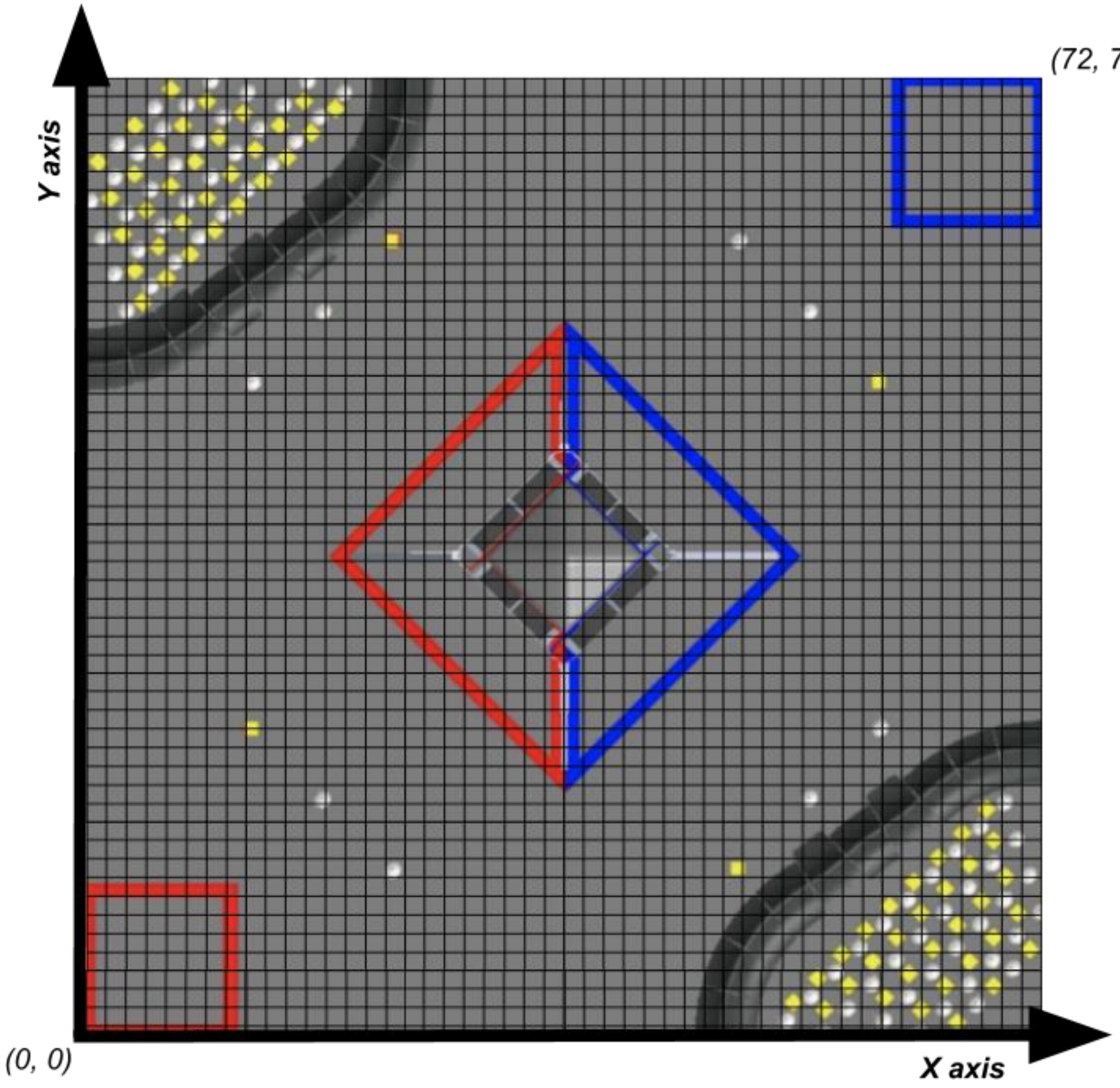
Due to the importance of the two positioning systems described below in our programming structures, we have affectionately coined the term MOE Position System, or MOEPS, to describe the systems.

Many of the following approaches and techniques we use rely upon a grid of (x, y) points. To form this grid, we divided up the field into a grid on the Cartesian plane, from (0, 0) to (72, 72).

A real field is 12ft. x 12ft., and we divided up the field into a 72 x 72 grid, where each MOE unit = 2 inches.

12 feet = 144 inches = 72 MOE units

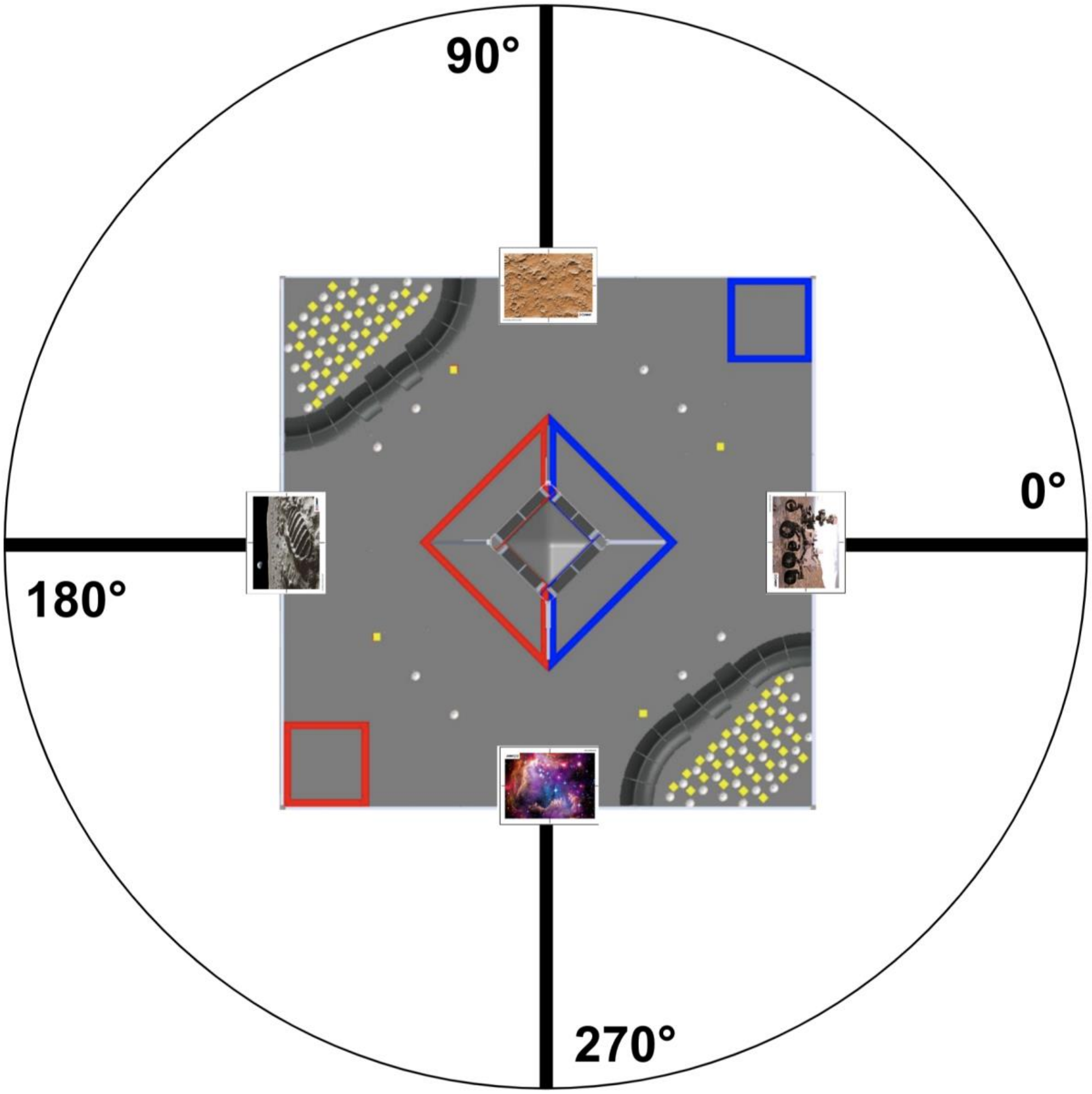
Additionally, the corner of the red depot was used as (0, 0) of the grid. Any of the four corners could have been used as (0, 0), so it was an arbitrary decision to choose the red depot.



To ease our programming, we made a Java class called PointMap to hold all important (x, y) points on the field with English names. While programming, we were able to refer to these names rather than the actual (x, y) coordinate. The following places were labelled as important:

Lander, Red Side Crater, Blue Side Crater, VuMarks, Field Corners, Sampling At Red Crater, Sampling At Blue Crater, Sampling At Red Depot, Sampling At Blue Depot, Red Depot, Blue Depot

Along with a positional (x, y) global map, we wanted to create an orientational global map to establish a consistent angle at any point on the map. The angle map was modeled off of the *Unit Circle*, which was used as a standard for marking angles on the map.

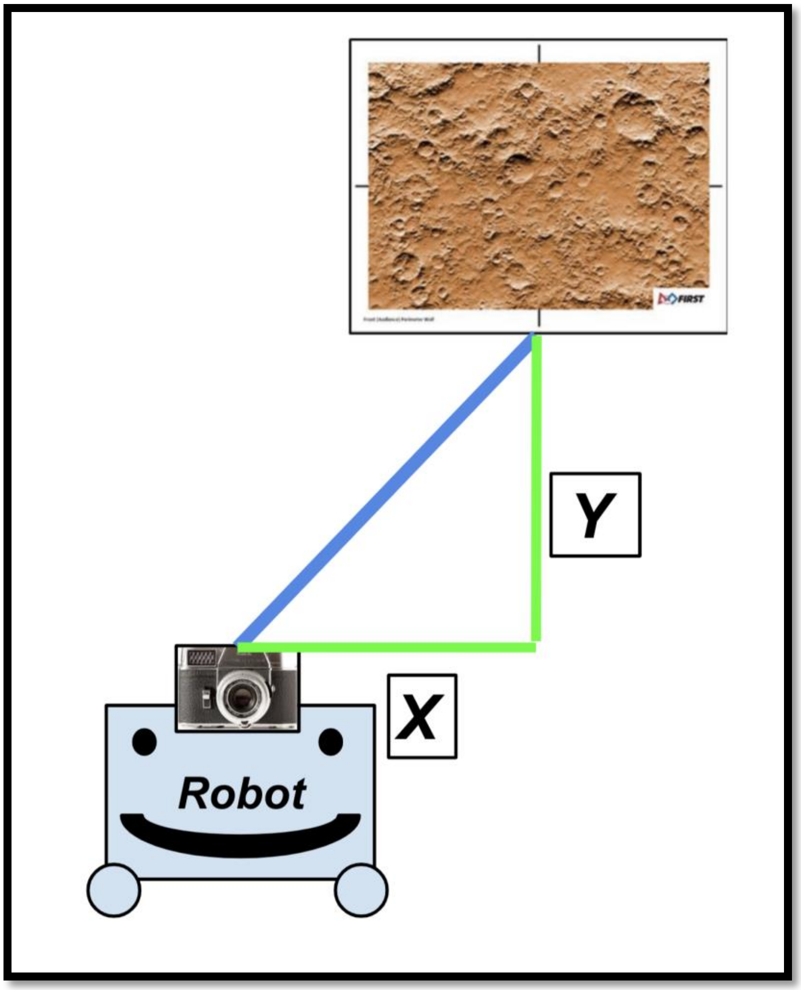

90°
0°
180°
270°

Localization

See engineering notebook entries: C42, C43, C146, C147, C148

In terms of our programming team, localization means to find out the robot's exact global (x, y) position on the field. In this case, the robot would have to find out its global (x, y) position on the **MOEPS global field grid** (check *Defining The Field Grid*). To accomplish this, we make use of the **Vuforia** image recognition technology and the **REV 2m Distance Sensor**.

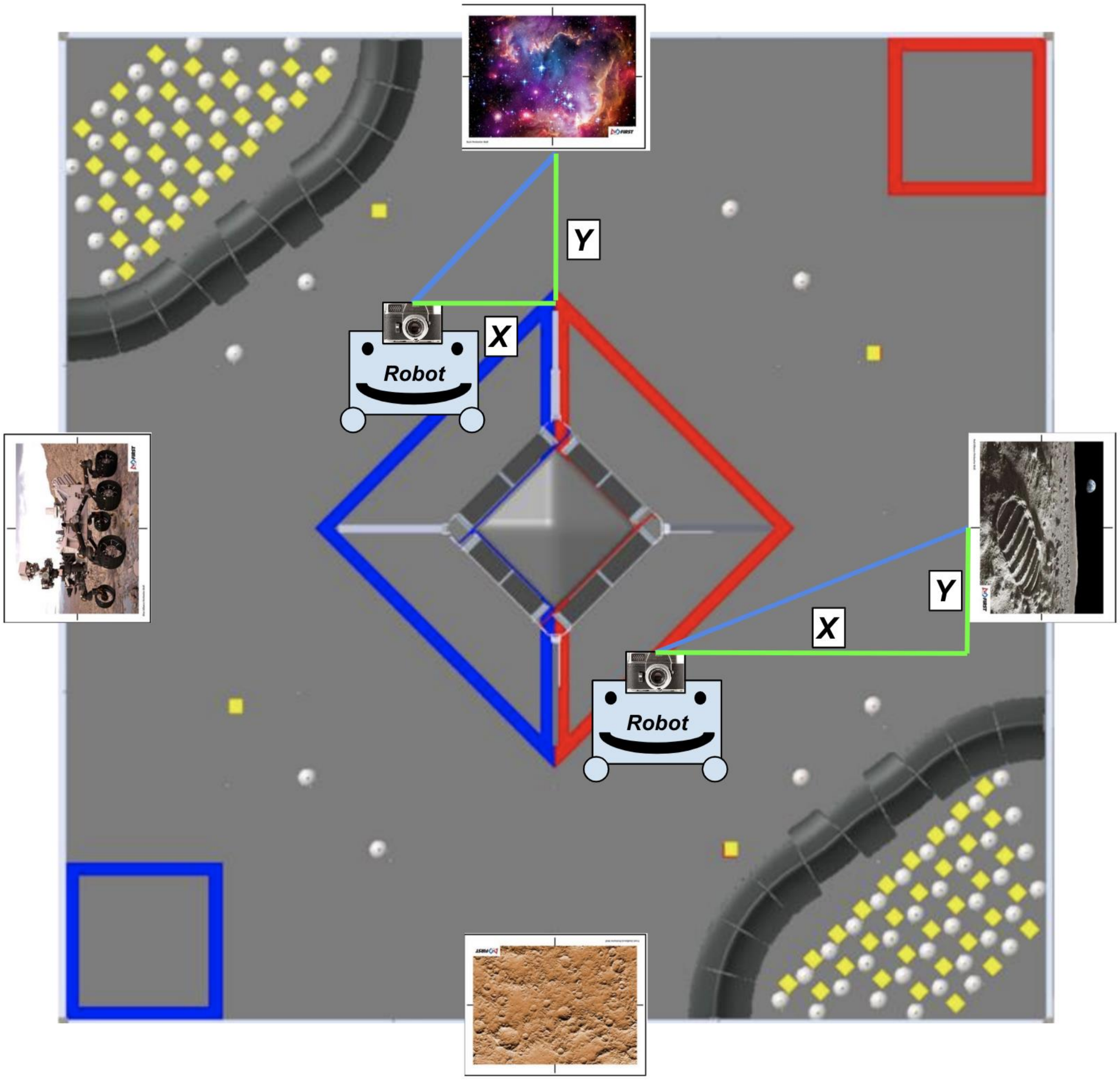
VuMark Localization



When the robot's webcam sees a VuMark, the following steps are taken:

1. Extrapolate horizontal (x) and vertical (y) distance from the VuMark using Vuforia
2. Scale the x, y distance into our 2-inch units – this is done by multiplying the values by the scalar 1/50
3. Depending on the VuMark, subtract the x, y values as appropriate – each VuMark has a distinct x, y position on the field, so subtract the robot's local x, y position from the VuMark from the VuMark's global position

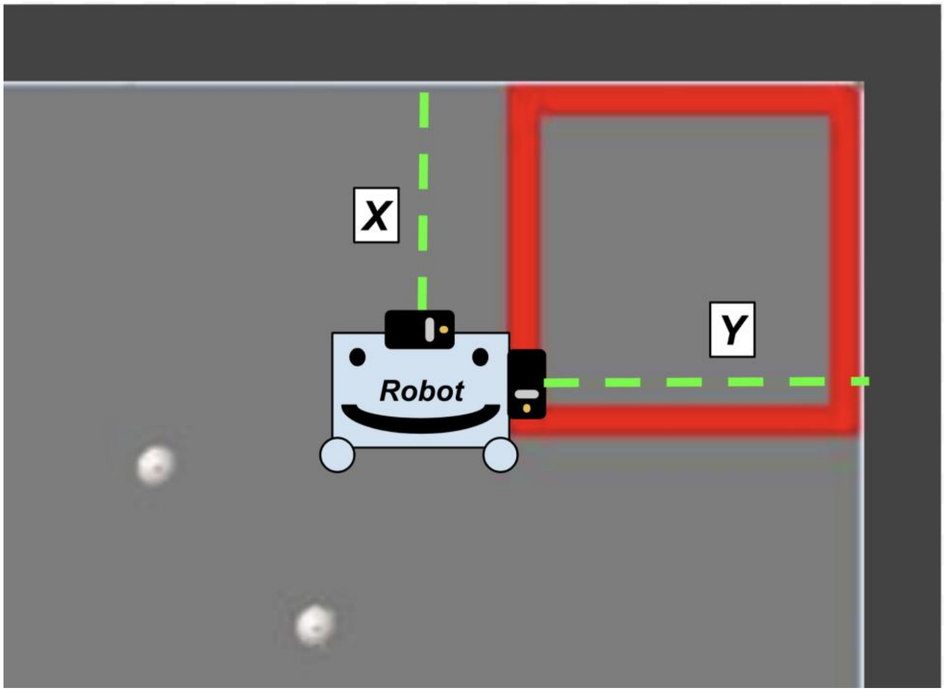
VuMark Localization on the field



Distance Sensor Localization

Localization using the distance sensor is similar to the VuMark Localization method, but is less reliable due to inaccuracies and errors that occasionally occur in the distance sensors. It used when no VuMarks are available and localization is necessary.

In this case, the following steps are taken: 1. Get vertical and horizontal distance in inches from wall with distance sensors 2.



Subtract inches from wall (x, y) position

3. Resulting (x, y) coordinate is the robot’s global location

Multithreading

Implementation of lift mechanism: C140

Implementation of realignment: C58, C59

Multithreading is a technique by which a single set of code can be used by several processors at different stages of execution. In other words, a program can have multiple sets of instructions running at the same time. With multithreading, the robot is able to do *more than one task* at any given time. Since our robot is trying to accomplish all standard autonomous points, time is often cut close to 30 seconds.

Without the use of multithreading, the robot's autonomous routine would have to speed up its motors significantly to meet the allotted 30 seconds. This speeding up results in less accuracy, resulting in an autonomous that is more prone to error.

Although processes like Vuforia and TensorFlow may run on separate threads, we intentionally use our own threads or pull from other threads for the following purposes:

- Bringing down the lift mechanism used for dropping/hanging
- Pathing algorithm realignment (see *Vuforia Listener* in *Additional Summary Information* for more details)

We also used **Atomic variables** for thread-safe operation. When a global variable is dealt with between 2 or more threads, there is always the danger of it leaking data when operations on it are done at the same time. Since using a raw variable without synchronization or any other standard is considered bad practice, we decided to use Atomic variables for thread-safe operation. This way, when communicating between the Main Robot thread and the Vuforia thread, we can guarantee that no strange behavior in autonomous occurs because of loss of data between the two threads.

Turning Methods

Conceptualization and implementation: C31, C32

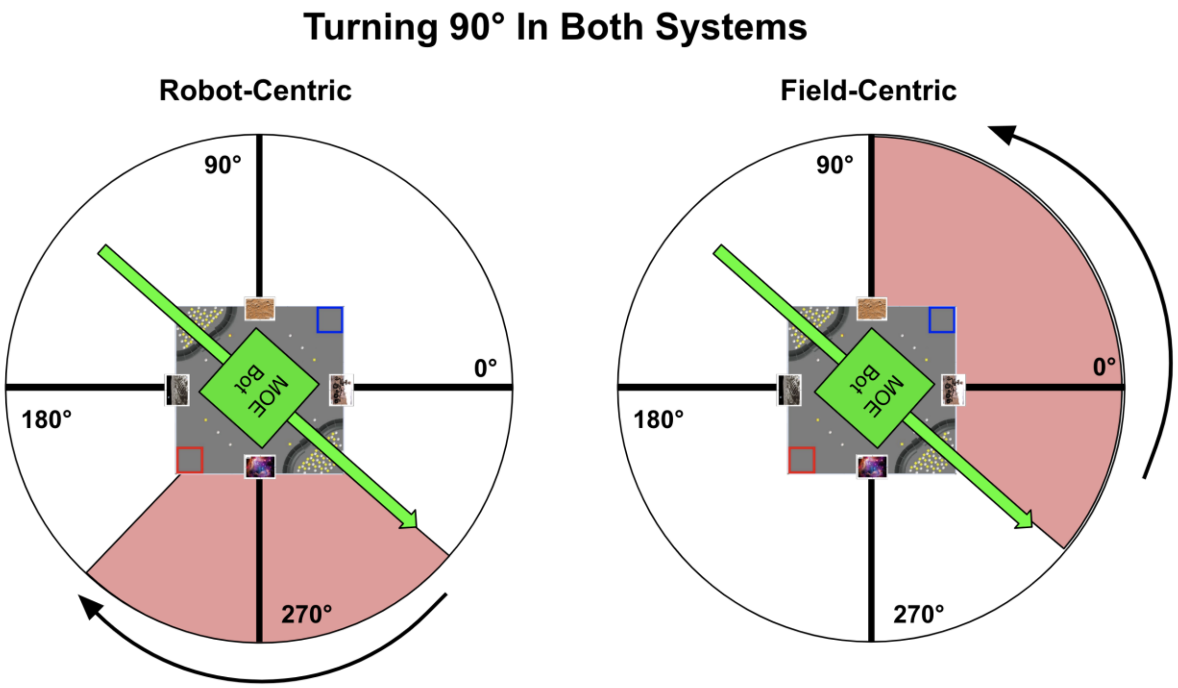
Turning in autonomous has to be precise to the degree for repeatable results, which is why turning is dictated by the **IMU** sensor built into the ***REV Expansion Hubs***. Instead of turning by time, we turn by setting the powers the motors and simply wait for the **IMU** to indicate that we are within the correct angle.

Field-Centric Turning & Robot-Centric Turning

In **field-centric turning,** the **MOEPS angle map** described above (see *Field Grid & MOEPS*), the robot turns to a given global angle on the field.

In **robot-centric turning** the robot turns to a given angle relative to itself.

Robot-centric vs. Field-centric Turning:



Turning 90° In Both Systems

Robot-Centric

Field-Centric

As shown in the diagram above, the robot turns 90° directly to the right in the robot-centric turn, while the robot is turning to the 90° mark in the field-centric turn. No matter the orientation, the robot will always turn to the same 90° mark in field-centric turning.

A* Pathfinding Algorithm & Dijkstra's Algorithm

*Conceptualization and implementation of old linear pathfinding algorithm (**not used on robot**):* C42, C43

Conceptualization and implementation: C46, C47

Implementation and testing: C50, C51

Debugging and gradual improvements: C55, C56, C58, C59, C64, C76, C93

Radius and size reductions: C97, C98, C99

2nd Stage Debugging: C104, C105

8-Directional Movement: C128, C129

3rd Stage Testing: C146

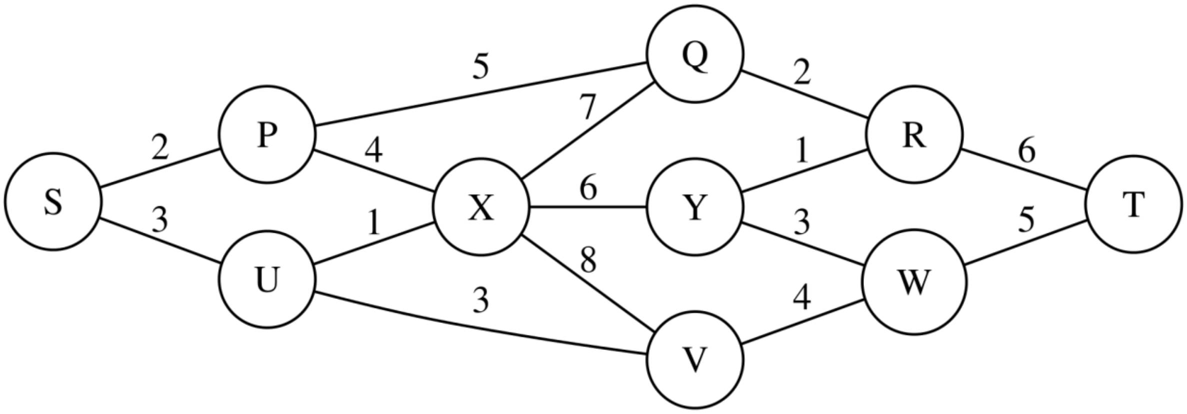
"Rotational Symmetry": C150, C151

Introduction

The purpose of the algorithm is to allow the robot to dynamically figure out how to reach its destination. In many autonomous pathings, whenever there is a slight disturbance, the autonomous fails to finish. Rather than explicitly giving the robot a path to follow, the robot is given an end destination. Through localization, the robot figures out its (x, y) coordinate on the field and calculates on its own how to reach the destination.

The *A* Pathfinding Algorithm (pronounced A Star)* is similar to the popular *Dijkstra's Algorithm*, which is used for finding the shortest paths between nodes in a graph. The only difference between the two is that the A* Algorithm utilizes a "heuristic function", or an approximation function, to approximate a faster solution to Dijkstra's algorithm. Dijkstra's algorithm checks many more cases than the A* Algorithm, therefore taking longer to arrive at a similar answer. Since the field we are using is 72x72 (5184) nodes, we wanted to guarantee that processing speed would be optimal in all situations and have two approaches available to use. The algorithms commonly deal with graphs shown like the one below, but had to be specially adapted in our case to work with a 2D grid.

Visual representation of traditional graph in computer science:



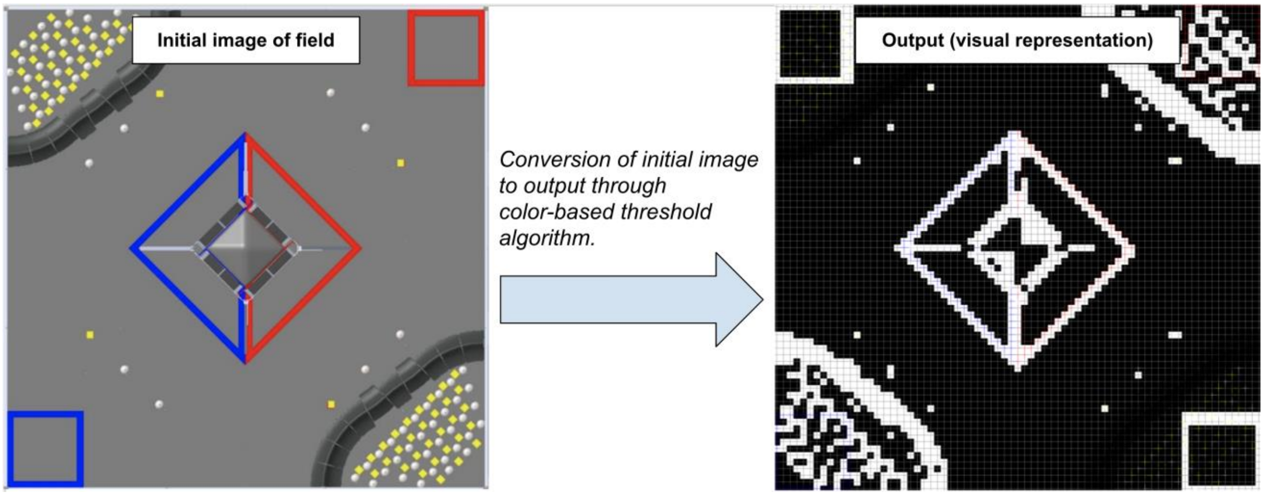
To account for processing speed & time, Dijkstra’s Algorithm has a worse case time complexity (when using lists) of **O(N^2)** where N = number of nodes on the graph, while A* has a time complexity of **O(b^d)**, where b = branching factor and d = depth of the solution on the search tree. Note that the time complexity of A* is worse when using a very expensive heuristic cost function, but we are using the simple Euclidean distance, or the distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

With the pathfinding algorithms, the robot is able to move in **8 directions** to go from point A to point B. The directions are labelled as follows: **North, Northeast, East, Southeast, South, Southwest, West, and Northwest.**

Setup

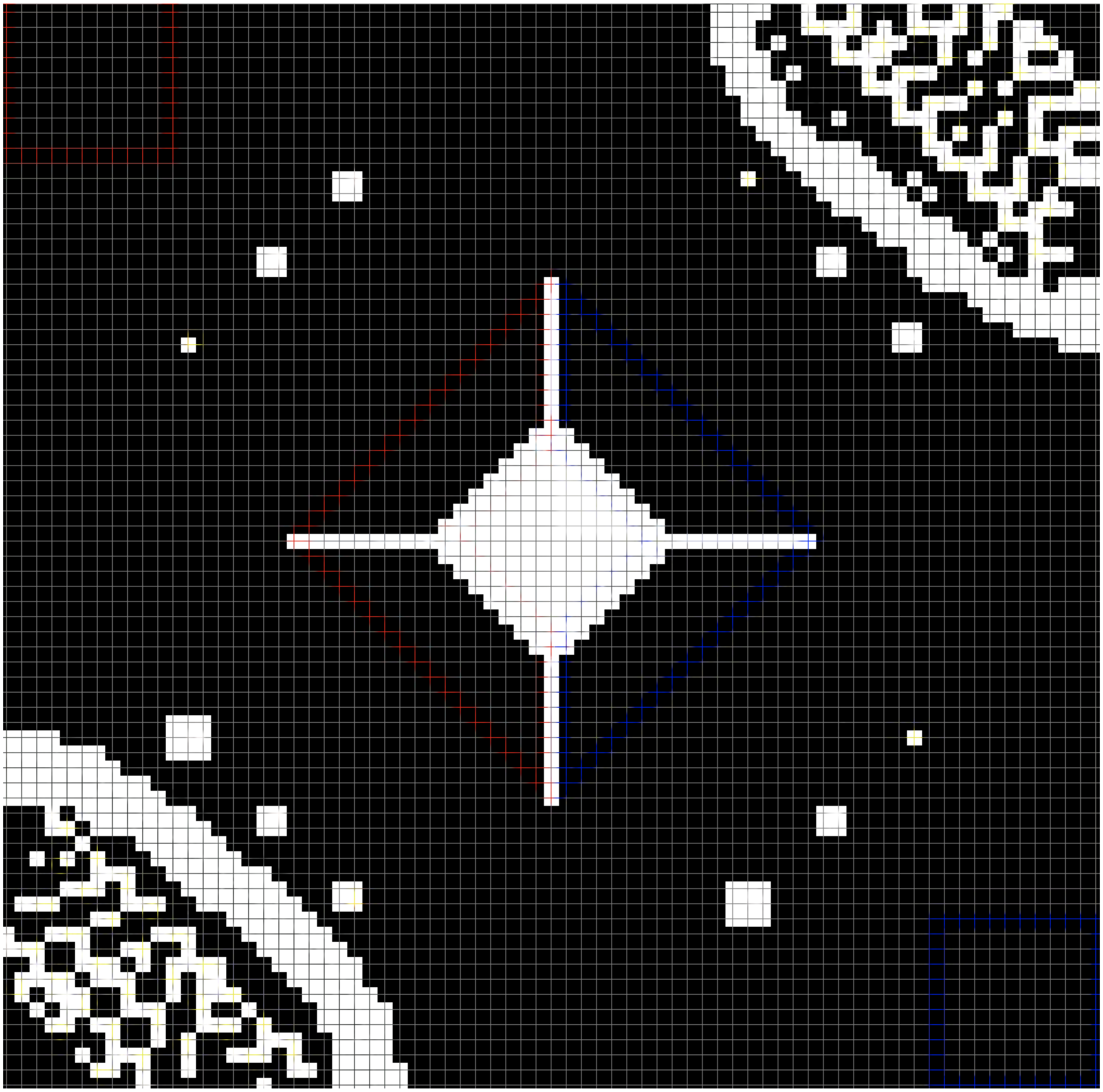
To utilize the **A* and Dijkstra algorithms**, we needed to first setup a graph. To accomplish this, we took a 2D image of the field from Game Manual 2. After that, we wrote a **Python** script (utilizing the *PIL imaging library*) to go through the image, converting it to points we deemed as barriers (white) and points we deemed as free space the robot could travel on (black). This conversion was done through a color-based threshold. In essence, the gray parts of the map were free space while the other colors were barriers. The output was an image with the converted points as well as a 72x72 *2-dimensional array* that we would be able to use as our graph for the A* algorithm. Also, the image was flipped because we wanted [0,0] of the 2D array to be the corner of the red depot, and [71,71] of the 2D array to be the corner of the blue depot.


Initial image of field
Conversion of initial image to output through color-based threshold algorithm.
Output (visual representation)

Mapped FTC Field (Visual Representation of Array)

1 (or white) = point a robot cannot travel on

0 (or black) = point a robot can travel on



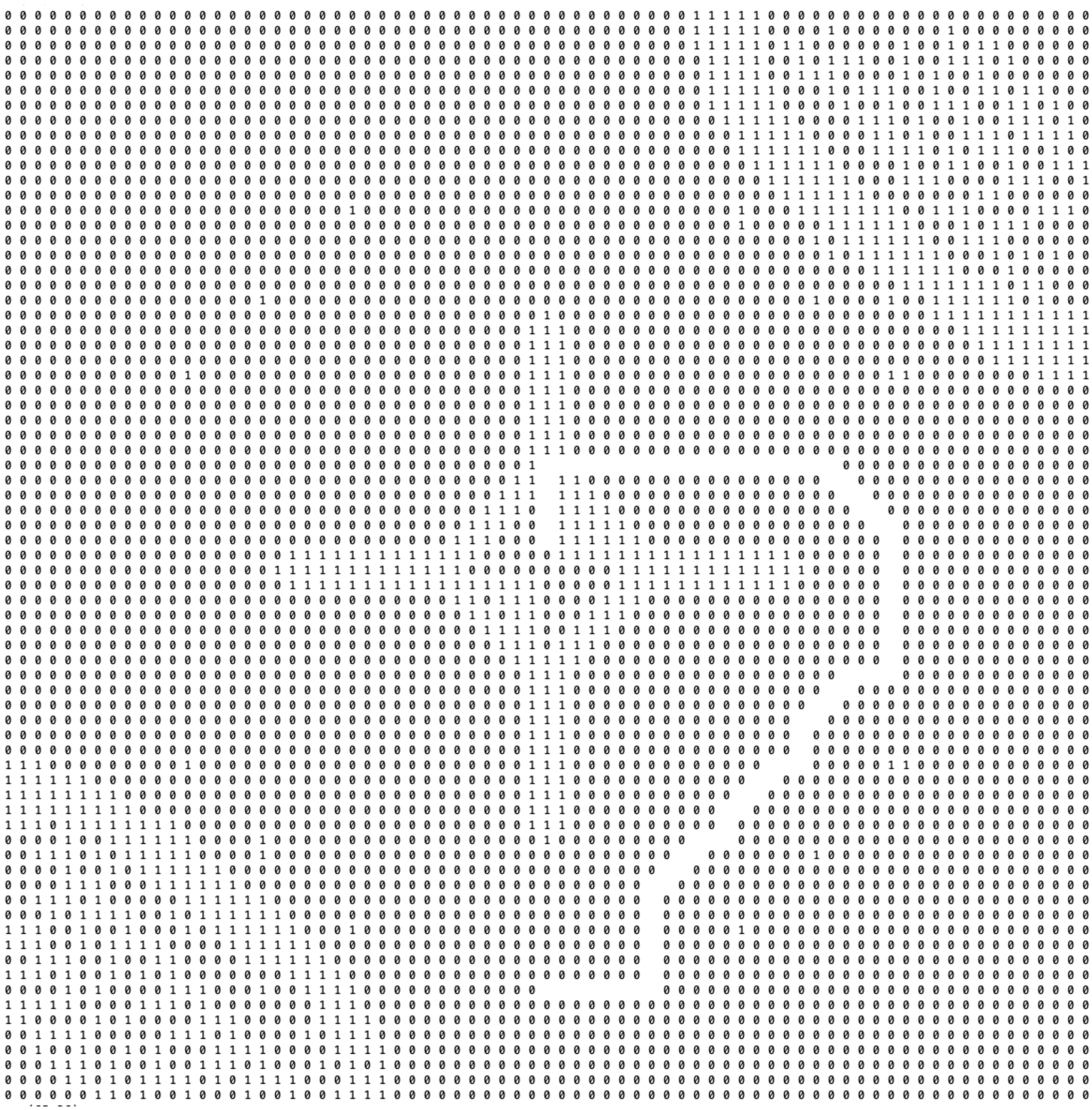
The original conversion had some errors, because places (depot, lines near the lander, etc...) were marked in white when they should have been open space. To fix this, we manually changed some values in the array. Since most of the conversion work was done by the Python script, this only took a few minutes.

The above image is what the output array looked like visually. The barriers shown in the image above are represented by 1s, while the free space shown in the image above are represented by 0s.

Implementation

We implemented the algorithm in Java, making a separate class to handle the calculations. To verify that we wrote the algorithm correctly and be able to make predictions on robot movements, we made a simulation to show the A* Algorithm's path from any Point A to Point B visually:

Initial Simulation

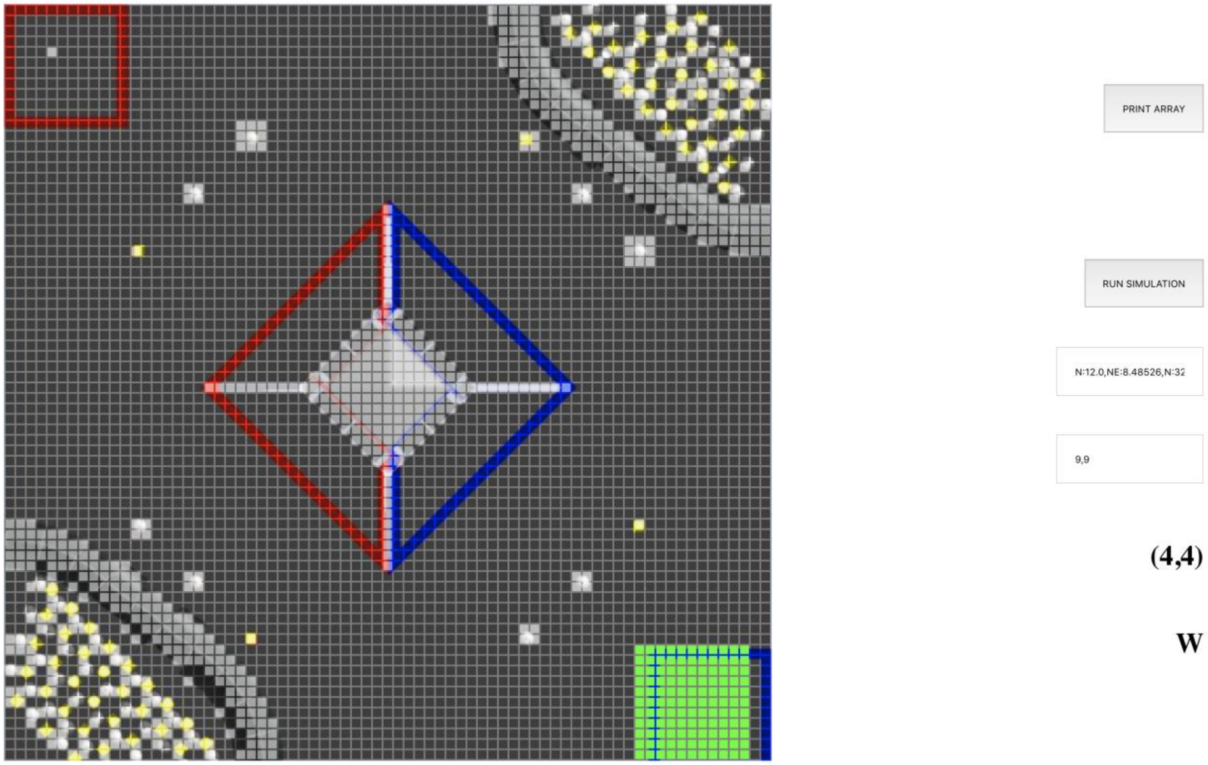


```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1                                         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1   1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1   1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0   1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0   1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0   1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0         0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0       0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0                 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

As complications with the algorithm increased, there was a need for a better simulation that more accurately depicted the algorithm in action. Below is a screenshot of an active simulation that shows the the robot (the green square), step by step, moving through the field.

Screenshot of Final Simulation

Video of live simulation: https://1drv.ms/v/s!AqOPfHs4_986ihlsFJLdiAYHtxG2



N:12.0,NE:8.48526,N:32

9,9

(4,4)

W

To now use the algorithm in practice, we had to convert the results into a usable format by writing an algorithm to do so.

Path Conversion Algorithm

(Input) - Original Pathfinding Results:	**(Output) - Usable Results:**
A series of points describing each point to go from point A to point B. *For example, getting from (0,0) to (5,5) could be:* (0,0) --> (0,1) --> (1,1) --> (2, 1) --> (3, 1) --> (3, 0) --> (4, 0) --> (4, 1) --> (5, 1) --> (5, 2) --> (5, 3) --> (5, 4) --> (5, 5)	The number of inches in each direction the robot has to go, in order (each unit is 2 inches). *For example, getting from (0,0) to (5,5) could be:* FORWARD 2 in. --> RIGHT 6 in. --> BACKWARD 2 in. --> RIGHT 2 in. --> FORWARD 2 in. --> RIGHT 2 in. --> FORWARD 8 in.

The robot first turns towards the **90° mark** described in the **MOEPS global angle map (**the direction facing the Crater/Mars VuMark). The results from the pathfinding algorithm would then be translated into movements for the robot based on encoder ticks. The result of this extensive process is a robust and repeatable movement system that allows the robot to figure out its own path when given two points on the field. This simplifies the process of adjusting and programming autonomous, as well as allowing for a more robust and dynamic movement system.

Pathfinding Algorithm Error Correction

The pathfinding algorithm worked perfectly well in theory, but in practice, there were a few issues we had to fix in order of importance.

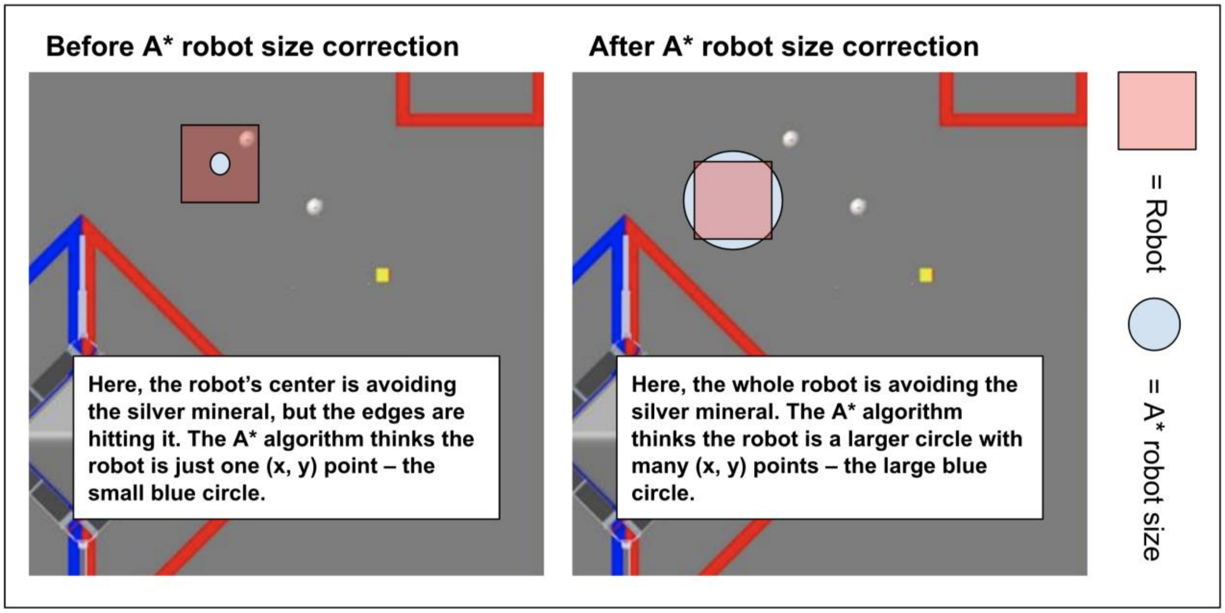
1. The **A* algorithm** treated the robot as a single (x, y) point on the **global 72x72 grid** while the robot actually comprised at least a circle of many points with radius about 10 inches. This led to the edges of the robot crashing into parts of the field (lander, crater, sampling, etc...) while its center thought it was following the pathfinding algorithms as a single small point.
2. While moving, the robot would turn slightly off angle. It would be not exactly at its end destination due to slight turning while making up, down, left, and right movements.

Error #1 – Size Corrections

Conceptualization and implementation: C55, C97

Second iteration: C98, C99

To fix this error, we modified the size of the robot in the algorithms. Instead of treating the robot as a single point, we treated it as a collection of multiple points – when put together, these points would form the robot rather than one small point.


Before A* robot size correction
Here, the robot's center is avoiding the silver mineral, but the edges are hitting it. The A* algorithm thinks the robot is just one (x, y) point – the small blue circle.
After A* robot size correction
Here, the whole robot is avoiding the silver mineral. The A* algorithm thinks the robot is a larger circle with many (x, y) points – the large blue circle.
= Robot
= A* robot size

Error #2 – Turn Corrections

To fix this error, we took the IMU sensor's horizontal angle before the robot followed the A* algorithm. We then constantly tracked the gyro sensor's angle while the robot followed the A* algorithm.

If the IMU's angle strayed by more than 2°, the robot self-corrected itself back to the correct angle by again utilizing the gyro to turn back into position.

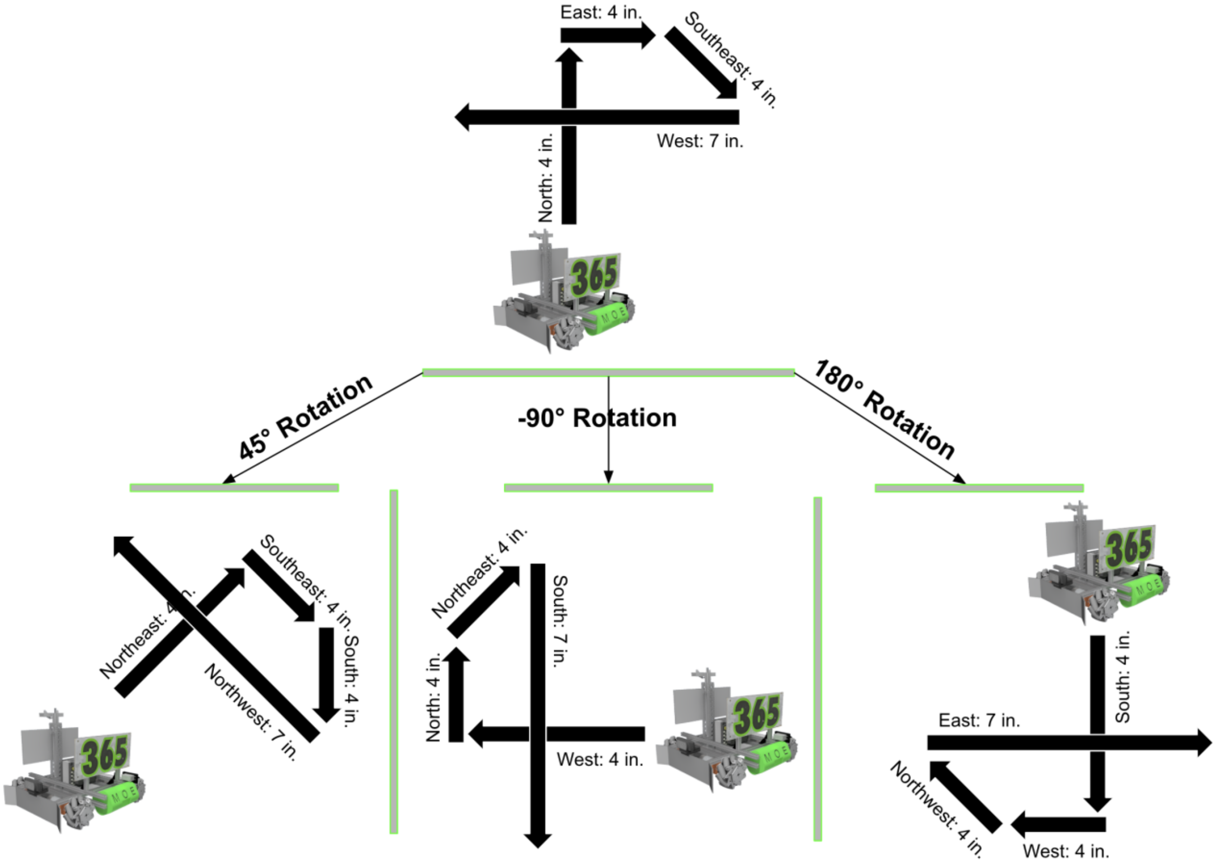
"Rotational Symmetry"

Conceptualization and Implementation: C150, C151

Another feature that we added to the pathfinding algorithm is the idea of "rotational symmetry". In other words, a given set of output instructions can be rotated by certain number of degrees while still preserving the relative directions of each movement.

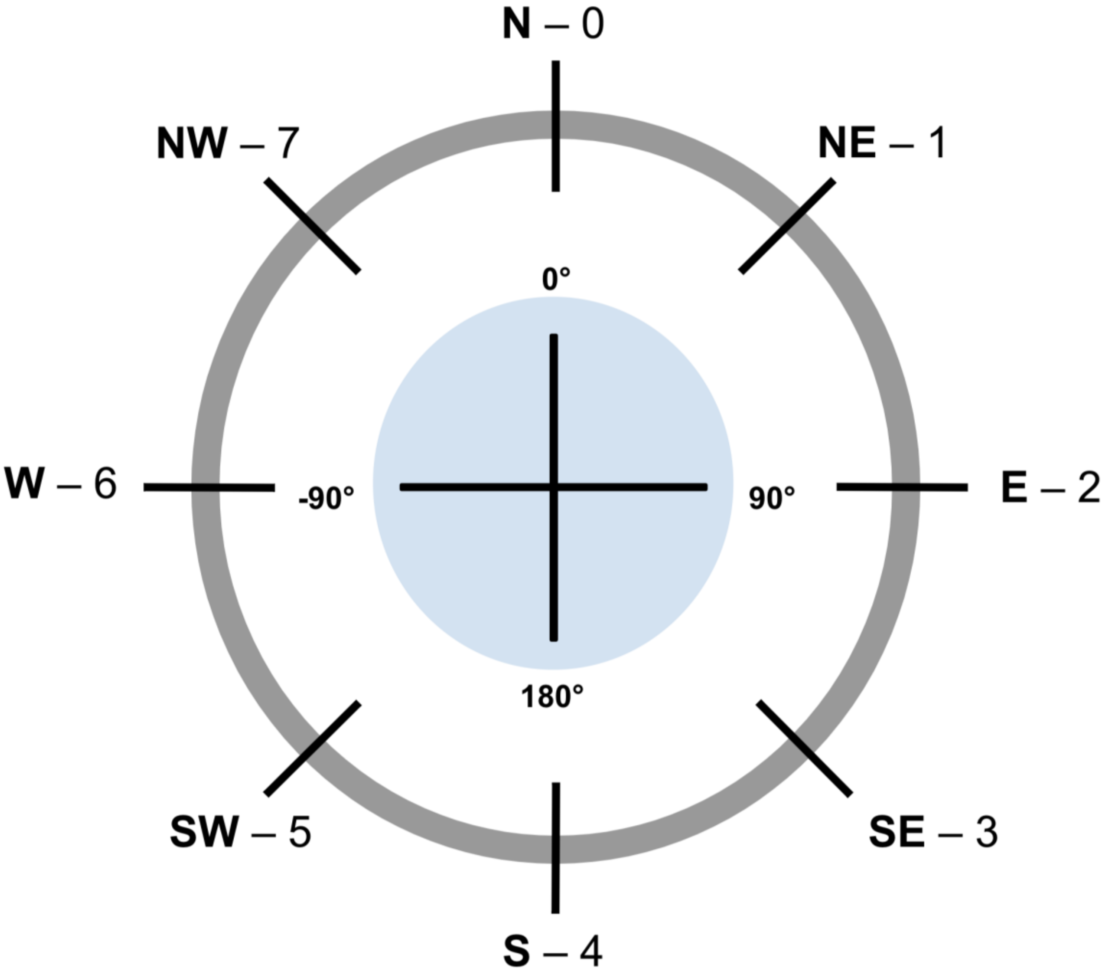
The purpose of rotational symmetry is to allow for optimizations in accuracy and speed of robot movements. Since moving forwards and backwards is always faster than strafing, rotational symmetry allows the robot to take a pathing from the Pathfinding Algorithms and rotate the pathing instructions. The robot can then quickly turn and apply the rotated instructions to allow for more forwards and backwards movements, resulting in a more robust movement.

Examples of rotations done on set of output instructions:



The algorithm is accomplished by setting a numerical value to each of the directions in clockwise order. North all the way around to Northwest is numbered from 0 up to 7. This simple numbering pattern makes rotation much simpler than writing each direction's rotation to its right. To rotate a direction, divide the angle needed to rotate by 45° and add it to the number. In the case a value goes above 7, it is wrapped back around to start at 0 (ie: 9 becomes 2).

Visualization of Direction to Number mapping, along with degrees associated with rotations:



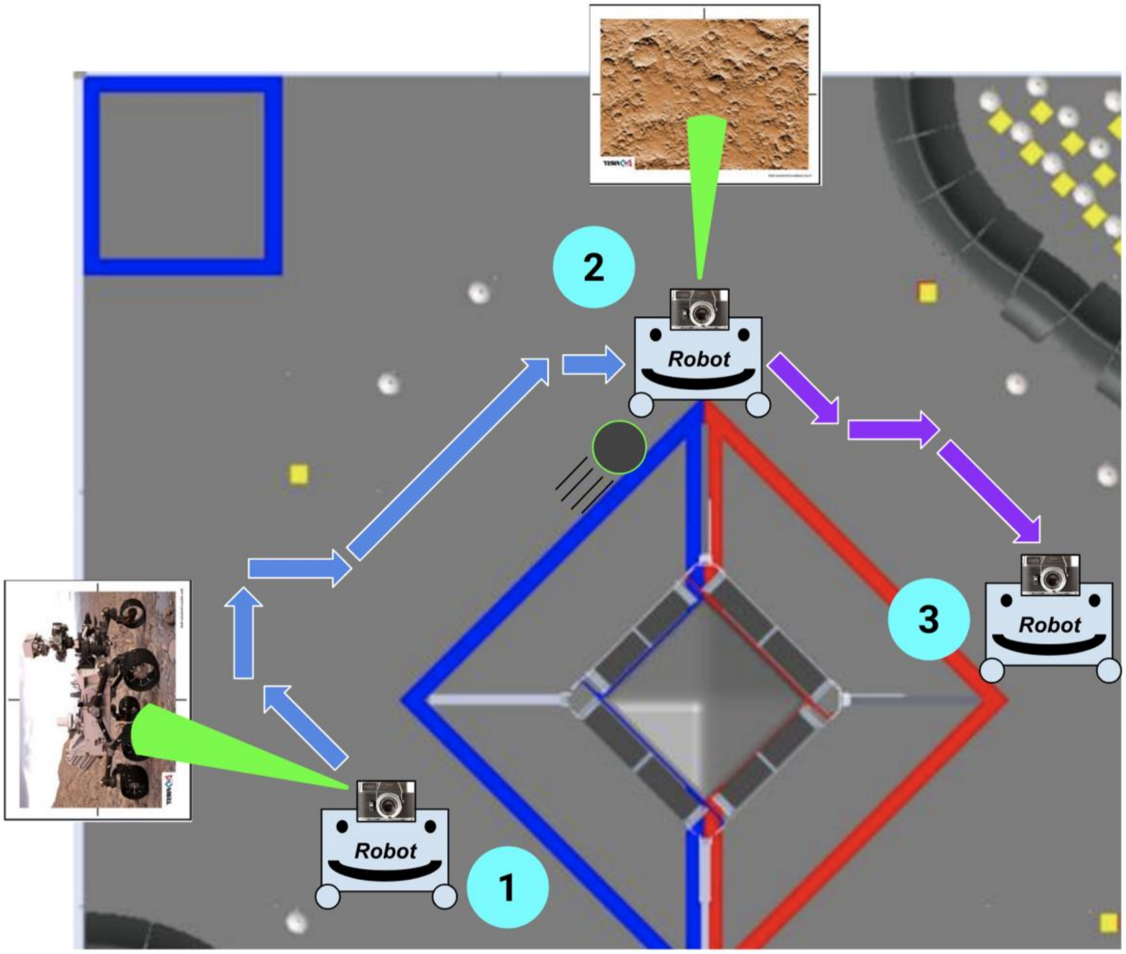
This system's simplicity becomes apparent when put into practice. For example, if an instruction says to move the robot North, the direction North's numerical value is 0. To rotate it by 90° clockwise, divide 90° by 45°, which equals 2. The 2 is added to North's numerical value, which results in 0+2 = 2. The 2 corresponds to the East direction, which is exactly a 90° clockwise rotation from North.

Realignment

Conceptualization and Implementation: C58, C59

As there is always a chance for error, such as another robot or debris in the way of a robot, a given robot might be knocked out of its planned path. This is another application for the Pathfinding Algorithms. During the course of following a path from the algorithm, the robot is always on the lookout for a new VuMark. (see *Vuforia Listener* in *Additional Summary Information*) If its camera sees one, it stops the current path it is on and restarts its pathing at the new VuMark. If the robot ever gets knocked off of its given path and fulfils the chance that it sees a new VuMark, the robot is able to get back on path.

The diagram below illustrates this process:



1. The robot localizes off of the Rover VuMark to figure out its (x, y) point
    a. Pathfinding Algorithms calculate a path to the destination (#3)
    b. Robot follows the pathing with encoders (blue arrows)
2. The robot is knocked off of its pathing by debris

   a. A new VuMark is seen and the robot stops its original pathing (blue arrows) and relocalizes, figuring out its new (x, y) point
   b. Pathfinding Algorithms calculate a path to the destination (#3)
   c. Robot follows the pathing with encoder (purple arrows)
3. Destination is reached

Error Correction & Fallback Plan B Routines

Throughout autonomous, there is opportunity for plenty of error to occur. Since a fundamental goal of autonomous is to have consistent, reproducible results, we try to better handle some errors that may result in a deviation from any planned autonomous route.

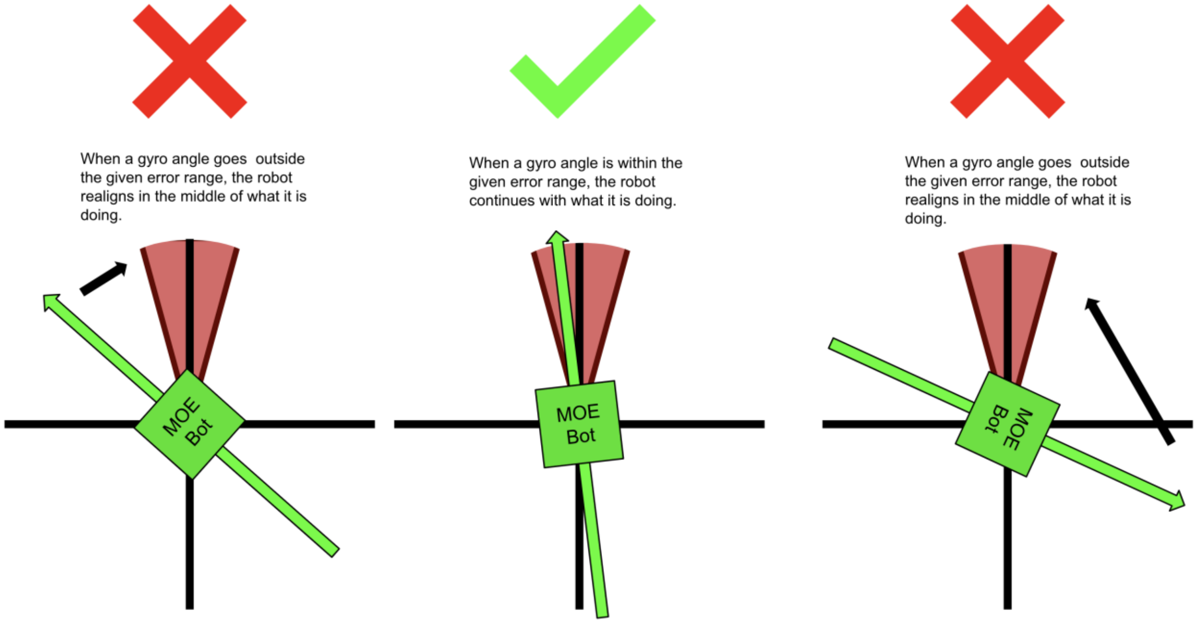
- Turn Corrections
- Distance Sensor Fallback

Turn Corrections

Implementation: C58, C59

In many of the routines and paths taken during autonomous, due to the nature of our mecanum wheels and the weight distribution of the robot, the robot gradually turns out of place. For example, when strafing normally for a period of 5 seconds, the robot could possibly turn 3° away from its starting angle. Over time, this error accumulates, and when sufficient, results in a faulty autonomous. To cut back on this, we define a given angle error range for any non-turning movement in autonomous. If the robot deviates from this error range, it interrupts what it is doing to turn back into proper position.

The diagram below shows the turn correction process.



When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.

When a gyro angle is within the given error range, the robot continues with what it is doing.

When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.

Distance Sensor Fallback

Due the occasionally unreliable nature of the distance sensors, there is a need for a fallback when they produce errors. When getting readings in the middle of autonomous, a distance

sensor sometimes gives wildly out of range readings or errors. To fix this, we add a fallback. If the sensor does not give reasonable values within 2 seconds, the robot uses a preprogrammed point to plug into the Pathfinding Algorithms rather than a more accurate point from the distance sensors.

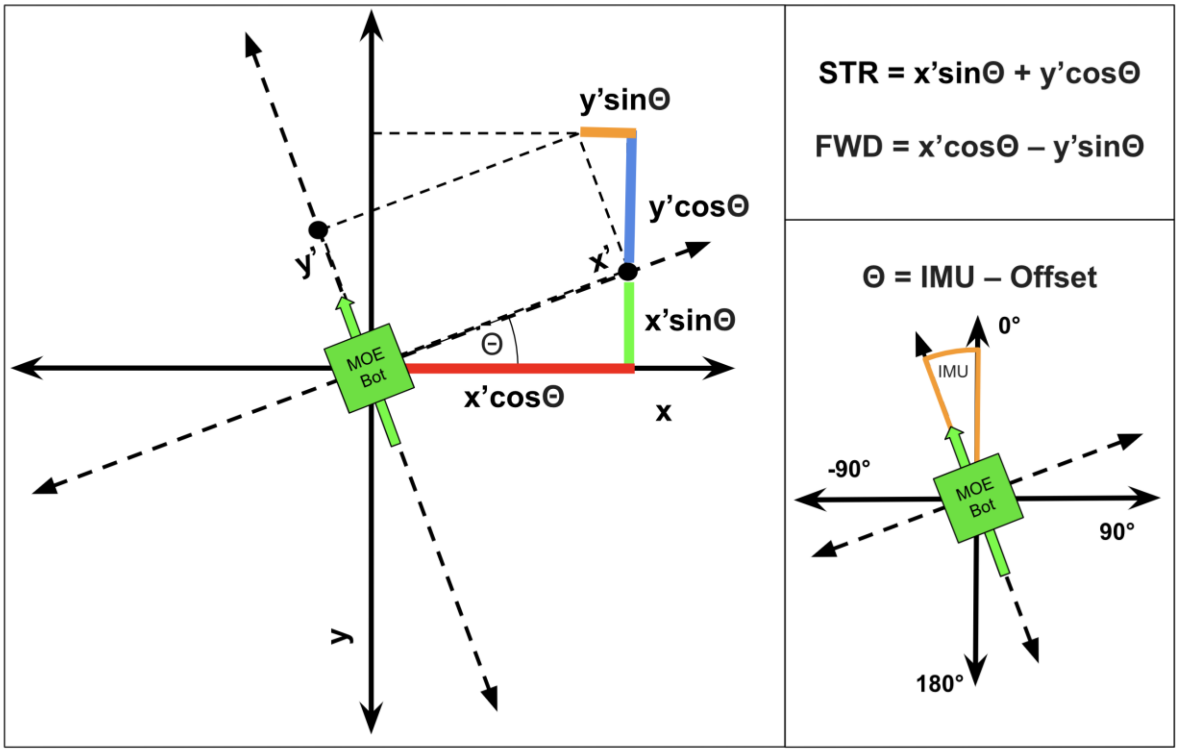
Driver Controlled Enhancements

Adjustable Field-Centric Movement

Conceptualization and implementation: C130, C131, C141

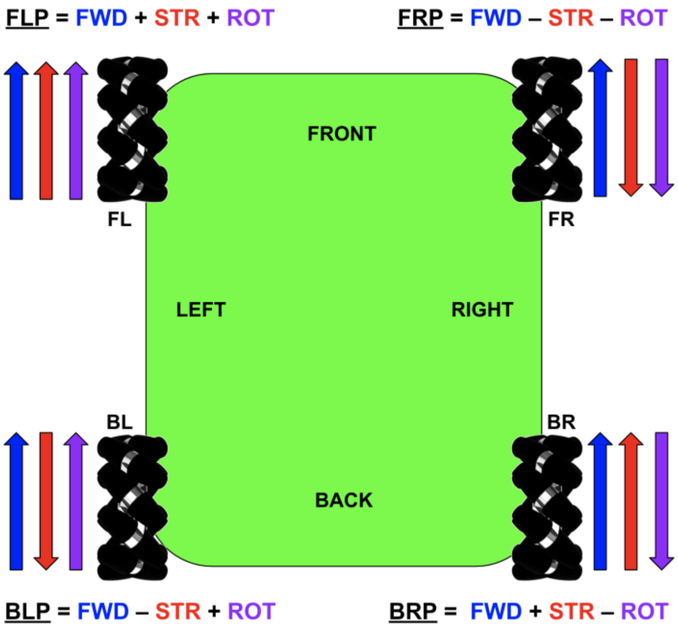
Since our robot uses a mecanum drive, it is capable of moving in any direction. Because of this, we are able to use **field-centric** motion rather than **robot-centric** motion. For the ease of the driver, we make all movements relative to the field rather than relative to the robot.

Note: In this diagram x' = Left Joystick X AND y' = Left Joystick Y



This diagram represents the **rotation of axes** that underlies the principles in field centric movement. The Θ is taken from the IMU, so that angle measurements are exact in the **field-centric** motion. Inputs for x' and y' are taken from the left joystick, and the right joystick x controls the rotation of the robot. Also, the offset for the IMU is to allow the driver to set a custom 0° point for the robot.

In the below diagram, the arrows indicate which direction the wheel turns when given a positive value for FWD (forward), STR (strafe), or ROT (rotation).



The corresponding values of FLP, FRP, BLP, and BRP are fed into the motors based on controller input so that field-centric movement occurs.

Lander Based Movement

Conceptualization and implementation: C141

Our robot also has “**lander specialized”** movement that allows for fine turning before hanging the robot. The D-pad allows for smaller adjustments in four-directional movement corresponding to the four buttons on the D-pad. Also, the bumpers allow for smaller adjustments in turning to fix orientation before raising the lift to hang.

Controls

Gamepad 1

Left Joystick: Field-centric movement in all directions

Right Joystick: Rotational movement (turns)

A: Reset 0° point (forward heading) for field-centric movement

Left Bumper: Turn slowly left (see *Lander Based Movement*)

Right Bumper: Turn slowly right (see *Lander Based Movement*)

D-Pad: (see *Lander Based Movement*)

UP → Move slowly forward

DOWN → Move slowly backward

LEFT → Strafe slowly left

RIGHT → Strafe slowly right

Autonomous Routines

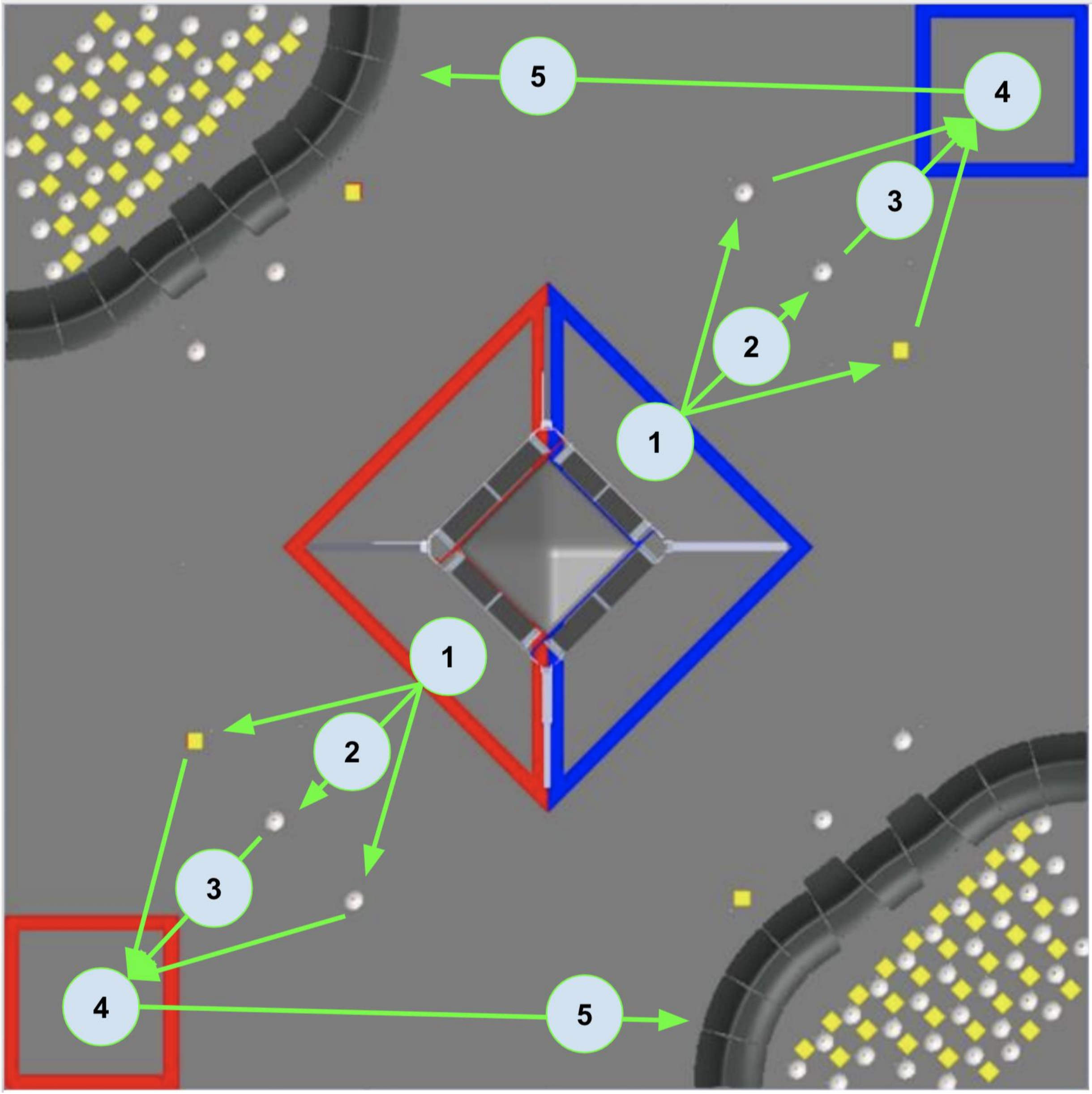
Conceptualization and testing of routine: C76

Iterative improvements, testing, and debugging: C84, C85

Testing environment fabrication: C104

2^{nd} *Stage Iterative improvements, testing, and debugging:* C105, C106, C121, C122

Testing and evaluations: C140, C141, C145



The image above represents our autonomous routine for the **depot starting points**. For the **crater starting points**, we only sample the gold mineral. We plan on executing the following steps for the depot starting points (the crater starting points are only the first two steps):

1. Landing & Detecting gold mineral
2. Travelling to and knocking off gold mineral
3. Travelling to depot
4. Deposit the Team Marker and leave gold mineral
5. Travel to crater & extend arm into crater

Initialization

1. Initialize motors, servos, sensors, and all other devices from the Hardware Map.
2. Initialize REV IMU sensor
3. Initialize Vuforia
4. Initialize TensorFlow
5. Reset Team Marker Servo to position 1
6. Reset Crater Extension Arm to position 1

1. Landing & Detecting gold mineral (30 pts.)

1. Use linear actuator to move lift up for given # of encoder tics
    a. The robot touches the floor and the claw goes above the top of the handle on the lander
2. Using multithreading, lower the lift back to its starting position
    a. (see *Multithreading* in *Key Algorithms*)
    b. The program continues on without waiting for the process to finish because of the multithreading
3. Turn ~70° to see 2 minerals and decide which is gold
    a. (see *Gold Mineral Detection Method* in *Additional Summary Information*)
4. The robot turns the appropriate # of degrees to face the gold mineral in left, right, or center

2. Landing & Detecting gold mineral (25 pts.)

1. Move forward appropriate # of inches to knock off gold mineral from its starting position
2. Continue moving forward to safely clear other 2 silver minerals

3. Travelling to depot

1. Turn appropriate number of degrees to face the depot
2. Move forward appropriate # of inches to reach the depot

   a. Note that the gold mineral is kept in control of the robot through the metal plate on its front – similar to a bulldozer

4. Deposit the Team Marker and leave gold mineral (17 pts.)

1. Turn appropriate # of degrees for front of robot to face the front/back wall
2. Localize and figure out (x, y) posi8tion on the MOEPS global field grid by using distance sensors
   a. (See *Distance Sensor Localization* in *Localization*)
   b. If the distance sensors have an error in measurement, fallback to Plan B
3. Turn appropriate number of degrees for left side of robot to face corner of field
4. Drop off Team Marker by setting Team Marker servo to position 0
5. Calculate path to crater using Pathfinding Algorithms
   a. ***Plan A:*** Use (x, y) position from distance sensors
   b. ***Plan B:*** Use default (x, y) position as an estimate – less accurate than Plan A
   c. (see *A* Pathfinding Algorithm & Dijkstra's Algorithm*)

5. Travel to crater & extend arm into crater (10 pts.)

1. Turn ~90° right for the back to face the crater
2. Apply 90° rotation to pathfinding algorithm results to fit new robot orientation
   a. (see *"Rotational Symmetry" in A* Pathfinding Algorithm & Dijkstra's Algorithm*)
3. Follow pathing to reach crater
4. Extend arm
5. Drive backwards to guarantee arm is in crater

Additional Summary Information

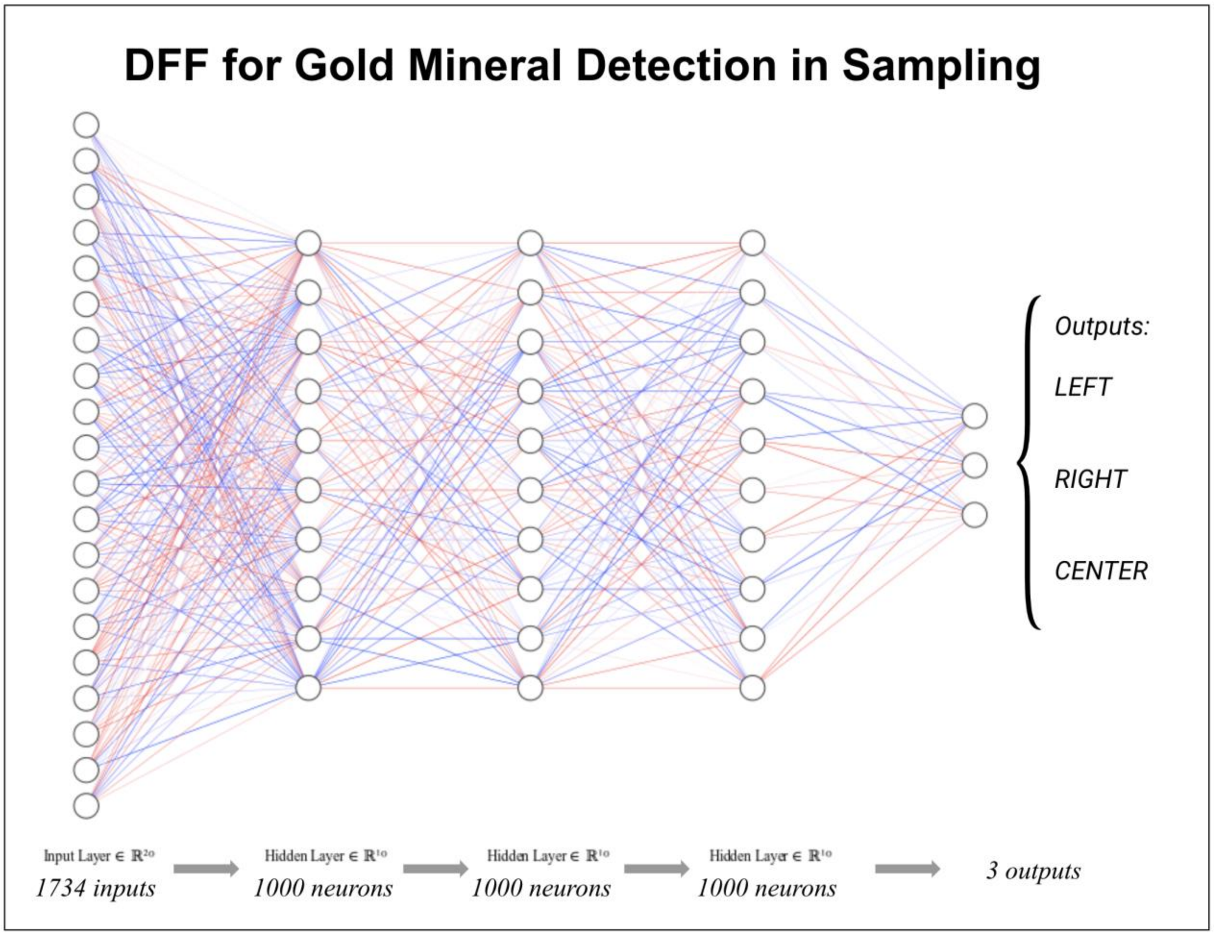
Creating An Artificial Neural Network (ANN)

Conceptualization and implementation: C110, C111, C112, C113, C114, C115

Before the integration of TensorFlow Lite into the official FTC App, we created a neural network with TensorFlow that could distinguish the left, center, or right position of the gold in the sampling minerals.

Choosing the Correct Structure

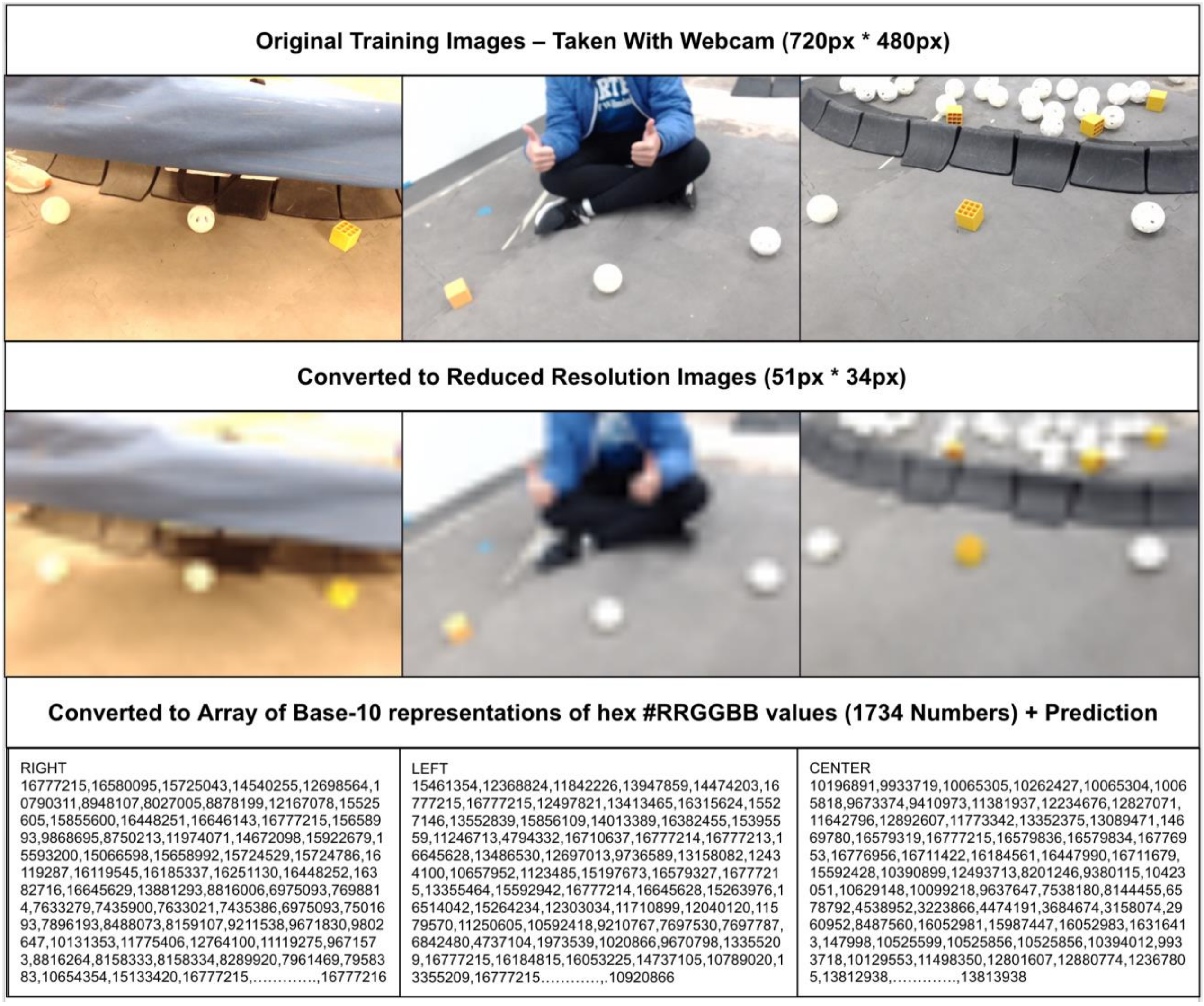
We decided to go with a ***deep feed-forward (DFF) neural network*** with ***backpropagation,*** a commonly used technique to train a neural network based around gradient descent.



There are 1734 inputs from the 1734 pixels in each of our input images, and 1000 neurons in each hidden layer. The final output has 3 possibilities. This neural network requires less training data because the problem at hand is fundamentally clear in terms of processing; there are no

complex edge detections required. All the neural network has to do is distinguish between yellow and white and their locations in the images.

Acquiring Training Data & Preprocessing (Total of 48 Images)



Original Training Images – Taken With Webcam (720px * 480px)

Converted to Reduced Resolution Images (51px * 34px)

Converted to Array of Base-10 representations of hex #RRGGBB values (1734 Numbers) + Prediction

```
RIGHT
16777215,16580095,15725043,14540255,12698564,1
0790311,8948107,8027005,8878199,12167078,15525
605,15855600,16448251,16646143,16777215,156589
93,9868695,8750213,11974071,14672098,15922679,1
5593200,15066598,15658992,15724529,15724786,16
119287,16119545,16185337,16251130,16448252,163
82716,16645629,13881293,8816006,6975093,769881
4,7633279,7435900,7633021,7435386,6975093,75016
93,7896193,8488073,8159107,9211538,9671830,9802
647,10131353,11775406,12764100,11119275,967157
3,8816264,8158333,8158334,8289920,7961469,79583
83,10654354,15133420,16777215,............,16777216
```

```
LEFT
15461354,12368824,11842226,13947859,14474203,16
777215,16777215,12497821,13413465,16315624,1552
7146,13552839,15856109,14013389,16382455,153955
59,11246713,4794332,16710637,16777214,16777213,1
6645628,13486530,12697013,9736589,13158082,1243
4100,10657952,1123485,15197673,16579327,1677721
5,13355464,15592942,16777214,16645628,15263976,1
6514042,15264234,12303034,11710899,12040120,115
79570,11250605,10592418,9210767,7697530,7697787,
6842480,4737104,1973539,1020866,9670798,1335520
9,16777215,16184815,16053225,14737105,10789020,1
3355209,16777215............,.10920866
```

```
CENTER
10196891,9933719,10065305,10262427,10065304,1006
5818,9673374,9410973,11381937,12234676,12827071,
11642796,12892607,11773342,13352375,13089471,146
69780,16579319,16777215,16579836,16579834,167769
53,16776956,16711422,16184561,16447990,16711679,
15592428,10390899,12493713,8201246,9380115,10423
051,10629148,10099218,9637647,7538180,8144455,65
78792,4538952,3223866,4474191,3684674,3158074,29
60952,8487560,16052981,15987447,16052983,1631641
3,147998,10525599,10525856,10525856,10394012,993
3718,10129553,11498350,12801607,12880774,1236780
5,13812938,............,13813938
```

Key ideas in images for the neural network:

- Changed background of images to show background does not matter
- Changed lighting of images to show lighting does not matter
- Changed tilt of images to show tilt does not matter
- Did not change order of minerals to show **only** ordering of minerals matter

NOTE: The preprocessing was automated using Python scripts to save time.

The reason the images' resolutions were reduced was due to the fact that training the network would require more time. Although training the network at full resolution would be fine, it

would possibly take a few minutes, and this can get cumbersome when refining and tweaking the data. We felt predictions could be made just as well at reduced resolution.

The process of converting to a Base-10 representation of hex #RRGGBB values:

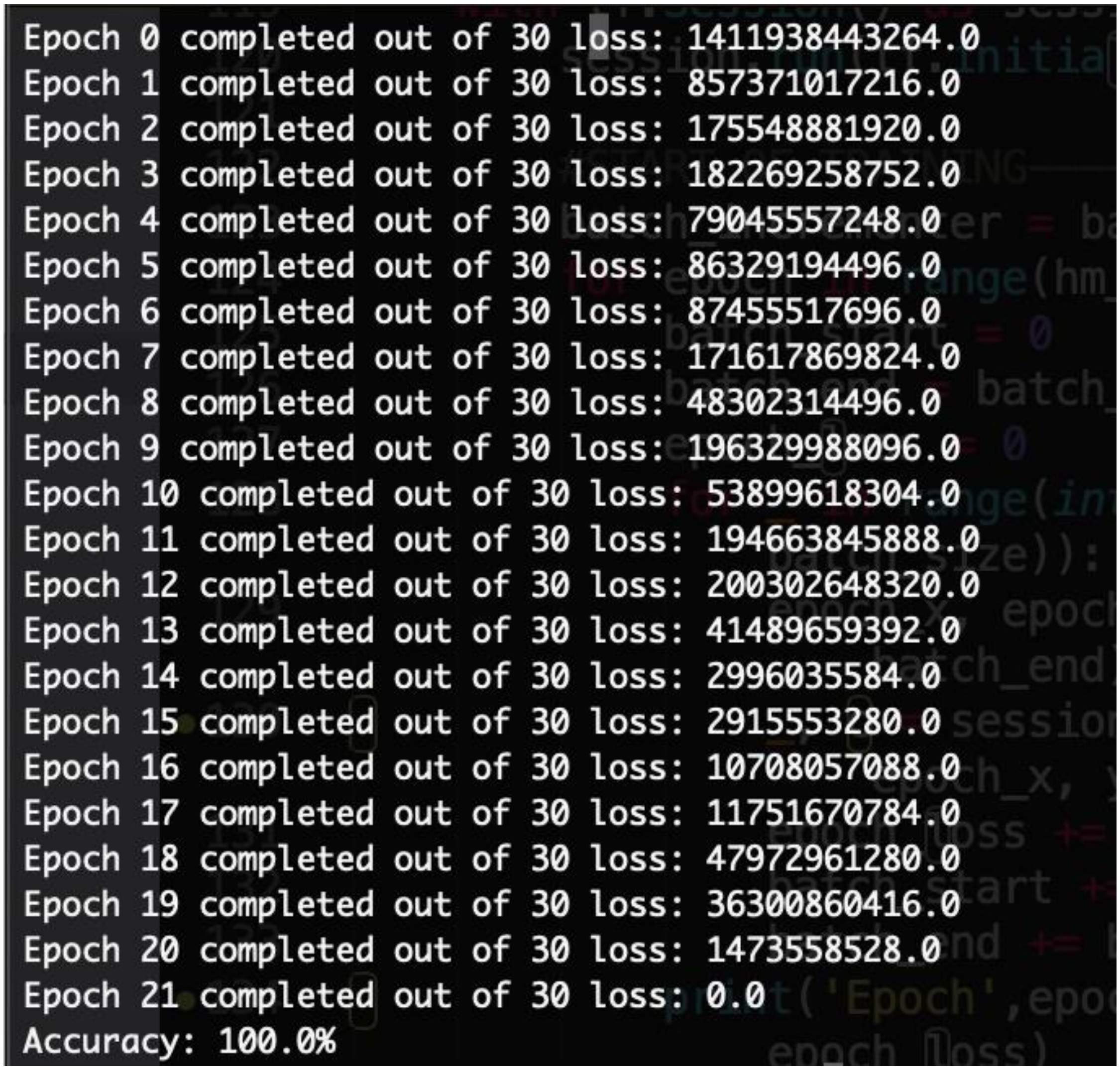
1. Take individual R, G, B (0 – 255) values of each pixel
2. Convert R, G, B values into one hexadecimal (base-16) number (#RRGGBB)
3. Take hexadecimal number #RRGGBB into a decimal (base-10) number

All the data was saved to a .txt file to be trained on later.

Training & Accuracy of Neural Network

Because we reduced the resolutions of the images in the preprocessing, training time for the 48 images was incredibly short: 5-15 seconds.

After training a successful model, here were our results.



```
Epoch 0 completed out of 30 loss: 1411938443264.0
Epoch 1 completed out of 30 loss: 857371017216.0
Epoch 2 completed out of 30 loss: 175548881920.0
Epoch 3 completed out of 30 loss: 182269258752.0
Epoch 4 completed out of 30 loss: 79045557248.0
Epoch 5 completed out of 30 loss: 86329194496.0
Epoch 6 completed out of 30 loss: 87455517696.0
Epoch 7 completed out of 30 loss: 171617869824.0
Epoch 8 completed out of 30 loss: 48302314496.0
Epoch 9 completed out of 30 loss: 196329988096.0
Epoch 10 completed out of 30 loss: 53899618304.0
Epoch 11 completed out of 30 loss: 194663845888.0
Epoch 12 completed out of 30 loss: 200302648320.0
Epoch 13 completed out of 30 loss: 41489659392.0
Epoch 14 completed out of 30 loss: 2996035584.0
Epoch 15 completed out of 30 loss: 2915553280.0
Epoch 16 completed out of 30 loss: 10708057088.0
Epoch 17 completed out of 30 loss: 11751670784.0
Epoch 18 completed out of 30 loss: 47972961280.0
Epoch 19 completed out of 30 loss: 36300860416.0
Epoch 20 completed out of 30 loss: 1473558528.0
Epoch 21 completed out of 30 loss: 0.0
Accuracy: 100.0%
```

As shown on the image, the accuracy is 100% on our 48 images. The loss of 0.0 might be an indication of over-fitting the training data, but the network was able to successfully predict our test data, so the network indicates it has not overfitted to the point of inaccuracy. In other words, when we gave the network new data, it was able to successfully determine whether the gold mineral was in the left, right, or center.

Gold Mineral Detection Method

Conceptualization and OpenCV: C63, C64, C70, C76, C77

OpenCV Testing: C84, C85, C105, C106

Official TensorFlow Model: C110, C111

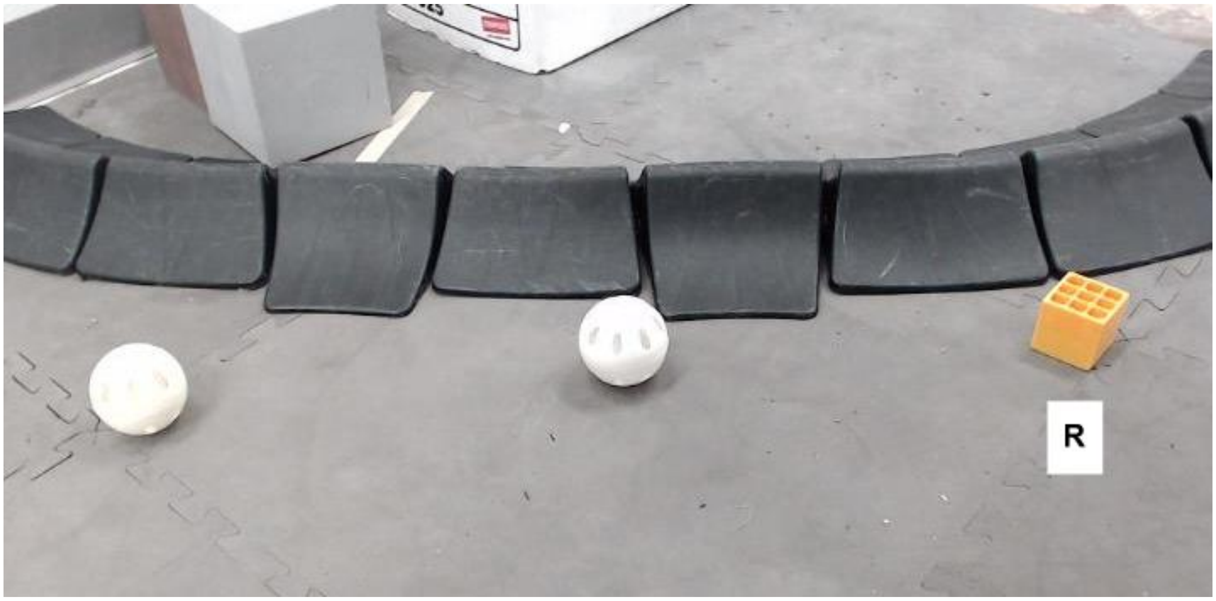
Custom neural network: C112, C113, C114

Final decision of official TensorFlow Model: C115

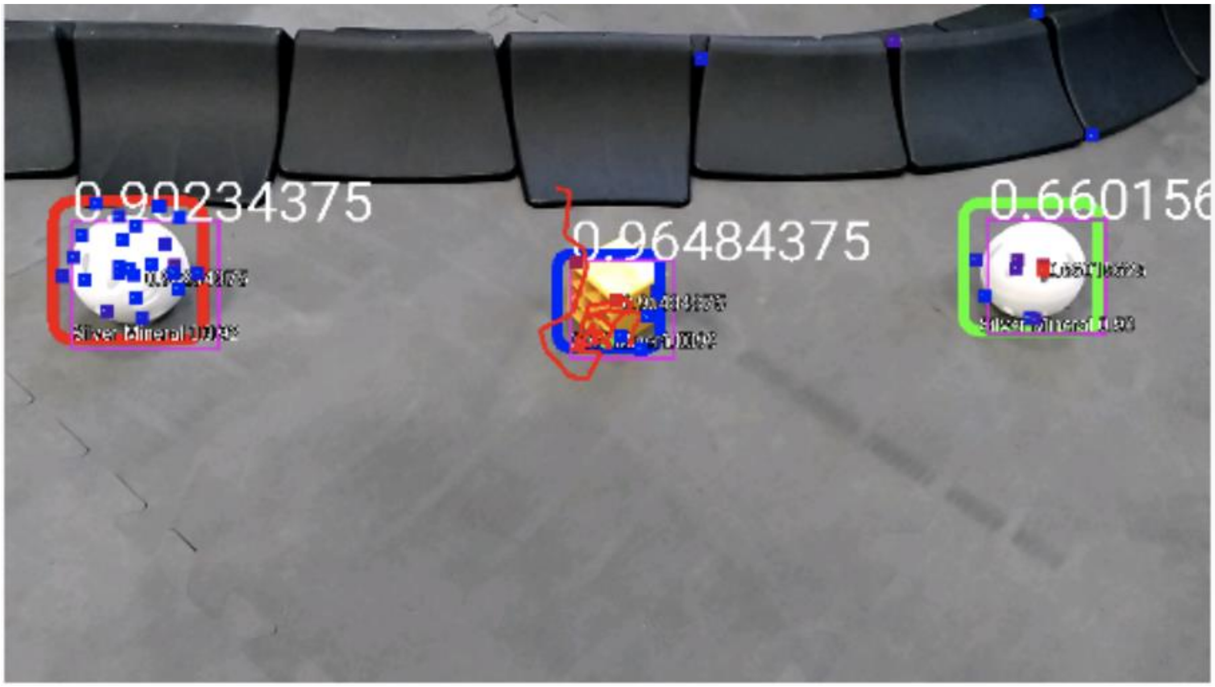
To finally end up with our implementation for the detection of the gold mineral, we went over 3 options: OpenCV, our own neural network, and the official TensorFlow Lite model for sampling. We felt that using color sensor would be too cumbersome given that we have a camera on the robot which we can do analysis on.

The reason we decided not to use OpenCV was because it could not run easily in parallel with Vuforia, and had too much overhead for the problem it was solving. We decided not to use our custom neural network because it did not have additional output like x position or estimated angle, unlike the offical TensorFlow neural network. In the end, we went with the official version for its robustness and availability of options.

Our TensorFlow Neural Network:



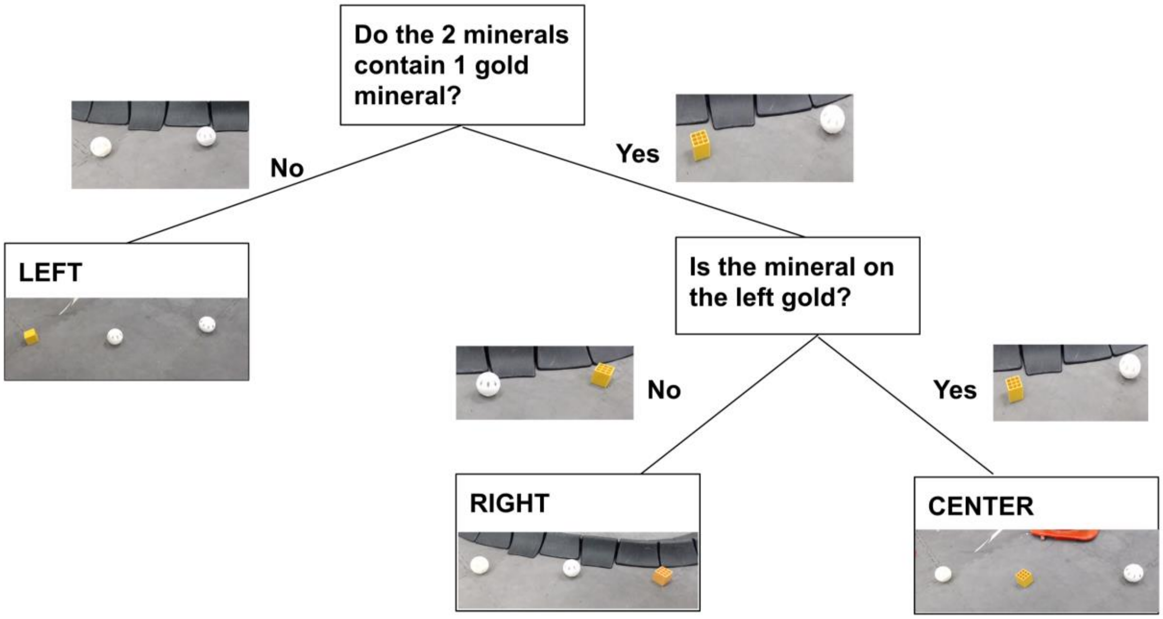
Official TensorFlow Neural Network:



Gold Mineral Decision Algorithm

See engineering notebook entries: C121, C122, C128, C129, C130

Since our robot was only able to see two minerals on the field, we had to write an algorithm to figure out which one is gold. To accomplish this, we guaranteed that the two minerals we saw would be the 2 right minerals out of the 3 in sampling. If the 2 were silver, the gold would be on the left. If the gold was the left of the 2 right minerals, it would be on the center. If the gold was on the right of the 2 right minerals, it would be on the right.


Do the 2 minerals contain 1 gold mineral?
No
Yes
LEFT
Is the mineral on the left gold?
No
Yes
RIGHT
CENTER

Vuforia Listener

(For the purpose of this, see *Realignment* in *A* Pathfinding Algorithm & Dijkstra's Algorithm*.)

To be able to retrieve the exact moment a Vuforia tag is seen by the camera, we took a unique approach in capturing these events. In Vuforia, there is a class known as the **VuforiaTrackableDefaultListener**. Normally, this class is called to check and retrieve a VuMark when one knows exactly when they will see a VuMark. However, due to the variable nature of our implementation of the A* and Dijkstra Pathfinding Algorithms (see *A* Pathfinding Algorithm & Dijkstra's Algorithm*), there is a need to know when a VuMark is found in a safe and efficient way.

This approach creates a class called **MOEListener** that extends the **VuforiaTrackableDefault Listener,** of which is used instead of the **VuforiaTrackableDefaultListener**. When overriding the methods in the default listener, we realized that there was no convenient method to realize when a *new* VuMark was found, so we modified one of the existing methods to let us know when a new VuMark was found. This way, in the event of a new VuMark, our robot would be notified properly.

Since Vuforia runs on a separate thread, there needed to be way to guarantee that the program would work consistently on multiple threads while being able to share information between the two. For this purpose, we utilized **Atomic** variables in Java for thread-safe sharing of information between the threads. If we did not, there is the slight chance that when two threads modify the same variable at the same time, there could be a loss of information. The usage of **Atomic** variables notifies the robot in the middle of following the path found by the Pathfinding Algorithms to stop what it is doing and realign against the VuMark.

Text-To-Speech (TTS)

For additional fun and utility, we incorporated the *Google Text-to-Speech* technology that allows text to be read aloud in a human-like fashion.

On the field, our robot likes to let us know how it is doing through a variety of phrases, including when it is initialized. Certain phrases include:

- “Initialized Vuforia”
- “Initialized Tensor Flow”
- “Initialization Ready”
- “Initialized Gyro”

Over time, hearing these phrases can become cumbersome. However, our robot likes to spice things up. When completing certain tasks in autonomous, the robot likes to show its patriotism and loyalty to our team. Certain phrases it uses include:

- “Go MOE”
- “Whooo!”
- “Hi _____” (where _____ may be someone’s name)
    - Note: this is pre-programmed, we have not yet integrated the facial recognition technology for the robot to detect people on its own

Our robot also has a fondness for music, which it may play on or off the field. More than anything else, our robot’s personality makes interacting with it more interesting and fun! :)

MOE, Miracles of Engineering

FTC Team 365

2018-19 Control Award Submission

Introduction

Throughout the design of our robot, we have kept one universal theme in mind.

User Friendliness: The measure of how robust, simple, easy to maintain, and easy to use a robot is.

To accomplish this goal of user friendliness in Autonomous and TeleOp, we have tried to keep the number of important components on the robot (sensors, motors, etc...) to a minimum while still vying to accomplishing our goals in mind. The result has been a robot that places more importance on intricate algorithms than sensors.

Our robot does utilize a good number of sensors, but wherever one can be omitted (for example: a camera rather than a color sensor), we take that option. This results in less environmental variables that can impact robot performance, as the robot relies on its algorithms and math to do computation in the place of sensors that could sometimes provide faulty data.

When driving the robot, we try to keep controls as simple as possible to allow the driver to focus on making important decisions rather than be distracted or bothered with the controlling of the robot.

Along with programming for the sake of the robot in competition, we have also programmed for the sake of learning (such as creating our own Neural Network!) to involve ourselves in other forms and kinds of programming. For an explanation on our thought process & more experimental procedures, view the Additional Summary Information. Also, below most titles will be a listing of notebook pages grouped together by what stage in the development process they show.

The 6 main sections are as follows:

1. Autonomous Objectives
2. Sensors Used
3. Key Algorithms
4. Driver Controlled Enhancements
5. Autonomous Program Diagrams
6. Additional Summary Information

Table of Contents

1.0 Autonomous Objectives

2.0 Sensors Used

3.0 Key Algorithms & Constructs

4.0 Driver Controlled Enhancements

5.0 Autonomous Routines

6.0 Additional Summary Information

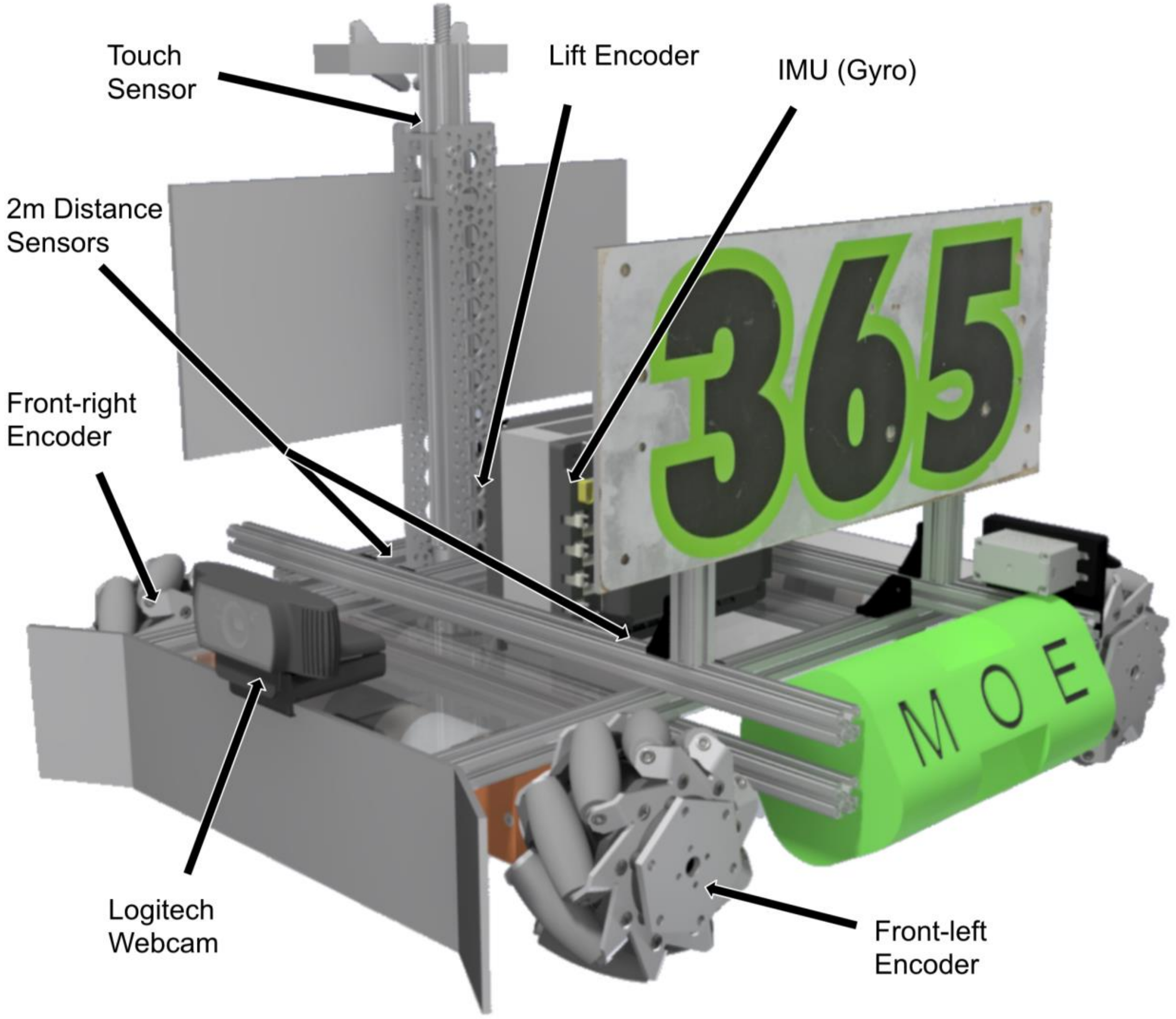
Autonomous Objectives

The following objectives are what we planned for in our robot's autonomous modes.

- **Autonomous Routine: (82 pts.)**
  - Landing – dropping off of the lander (30 pts.)
  - Sampling – knocking off the gold mineral (25 pts.)
  - Sampled gold mineral placed in depot (2 pts.)
  - Claiming – dropping the Team Marker in the depot (15 pts.)
  - Parking – ending autonomous in the crater (10 pts.)
- **Algorithmic & Programming Objectives:**
  - Establishing Field Grid & MOEPS (MOE Positioning System)
  - Localization
  - Multithreading
  - Accurate Turning Methods
  - Accurate Pathfinding with A* and Dijkstra's Algorithms
    - Pathfinding Error Correction
    - "Rotational Symmetry"
    - Realignment
  - Error Correction & Fallback Plan B Routines

Although not completely relevant in explaining the controls and actions of the robot, we included additional algorithms and details in our programming process that we felt to be of importance in the *Additional Summary Information*.

Sensors Used



Encoders (3)

2: Encoders placed on motors involved with the robot’s mecanum drive. One encoder is on the front-left wheel, while the other is on the front-right wheel.

1: Encoder placed on the lift motor involved with dropping/hanging. The encoder is used for precise, controlled motion of the lift motor.

Inertial Measurement Unit (IMU) (1)

1: IMU build into the REV Expansion Hub. This IMU is effectively a gyro sensor, with the capability to measure rotation on 3 axes. We primarily use the horizontal axis, or the one that measures rotation parallel to the ground for accuracy in any turns or rotational movement of the robot.

Logitech Webcam (1)

1: Logitech Webcam used in the front of the robot. Using this rather than a phone camera allows for the phone to be safely protected within the robot, making sure that nothing goes wrong during the match. The Logitech Webcam is used for recognizing the Vumarks.

REV 2m Distance Sensor (2)

1: REV 2m Distance Sensor placed on the front side of the robot, facing the forward direction. The sensor is primarily used for telling distance away from the walls of the field.

1: REV 2m Distance Sensor placed on the right side of the robot, facing the right direction. The sensor is primarily used for telling distance away from the walls of the field.

Touch Sensor (1)

1: Touch Sensor placed near the top of the Tetrix channel used for holding the linear actuator used in dropping/hanging. The sensor is used for safe and automated resetting of the lift in hanging. Since the lift has to have the same starting point across all robot autonomous modes, a standard starting point is important.

Key Algorithms & Constructs

Field Grid & MOEPS (MOE Positioning System)

Conceptualization and implementation: C7, C8, C13, C14

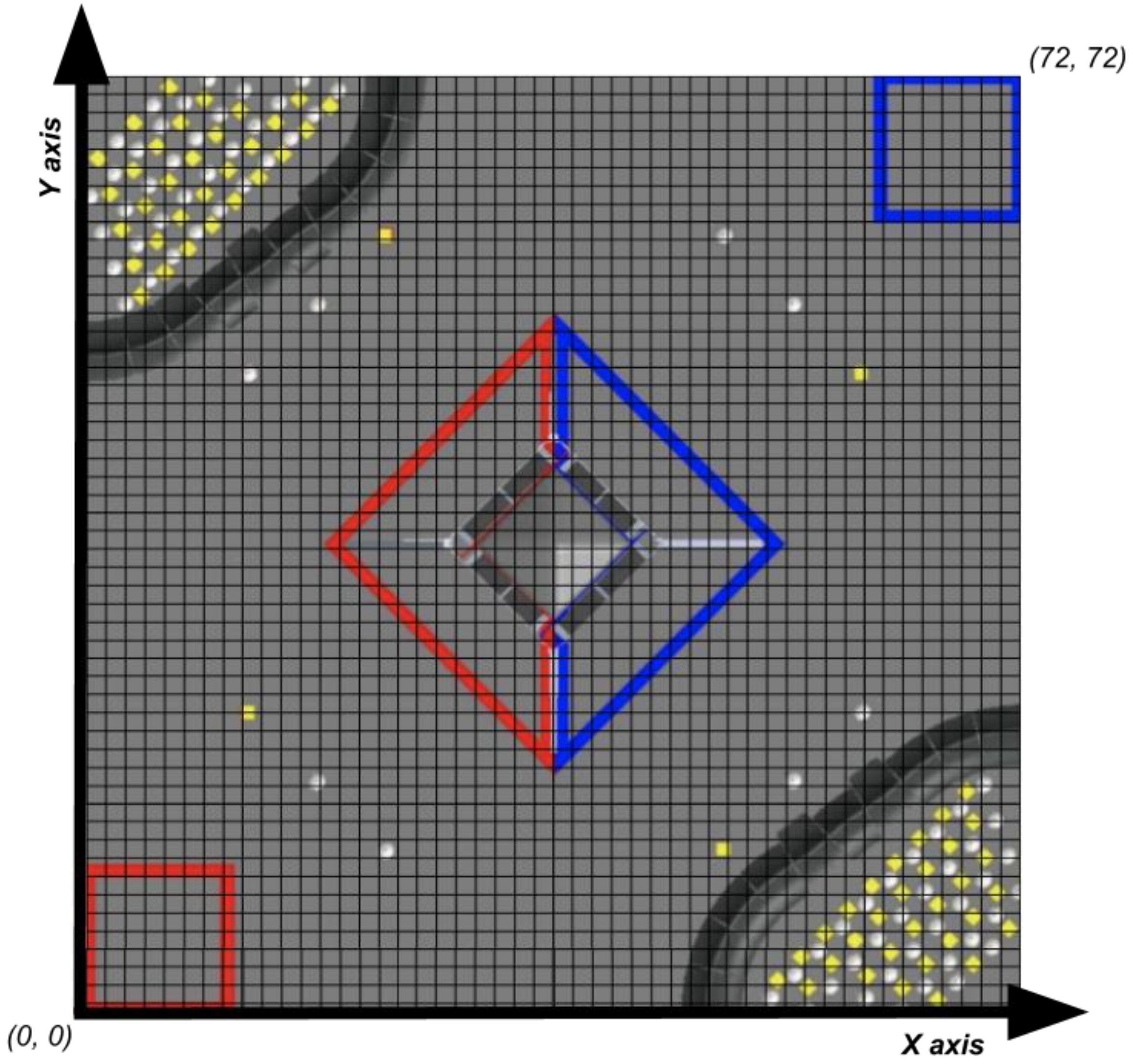
Due to the importance of the two positioning systems described below in our programming structures, we have affectionately coined the term MOE Position System, or MOEPS, to describe the systems.

Many of the following approaches and techniques we use rely upon a grid of (x, y) points. To form this grid, we divided up the field into a grid on the Cartesian plane, from (0, 0) to (72, 72).

A real field is 12ft. x 12ft., and we divided up the field into a 72 x 72 grid, where each MOE unit = 2 inches.

12 feet = 144 inches = 72 MOE units

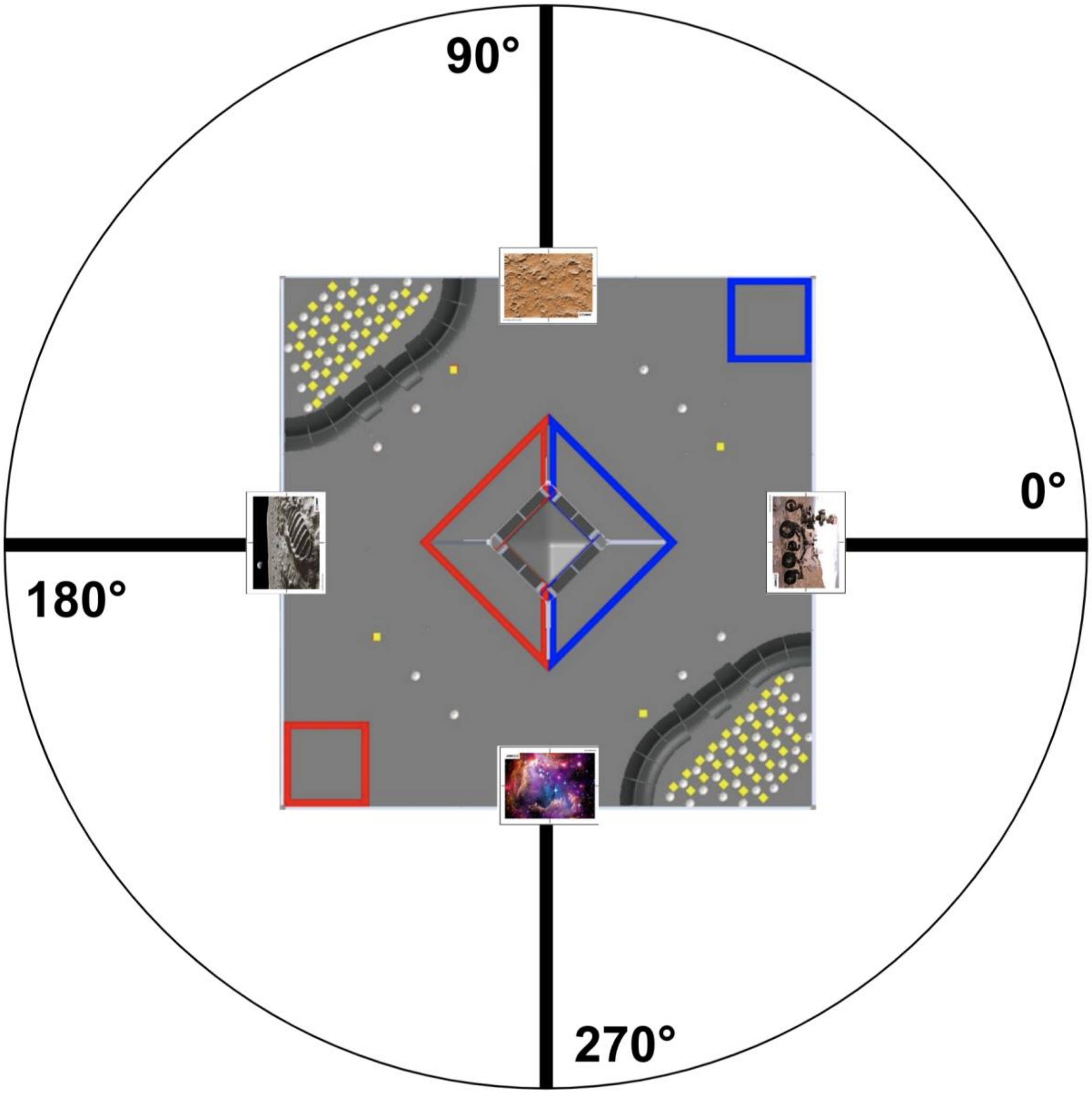
Additionally, the corner of the red depot was used as (0, 0) of the grid. Any of the four corners could have been used as (0, 0), so it was an arbitrary decision to choose the red depot.



To ease our programming, we made a Java class called PointMap to hold all important (x, y) points on the field with English names. While programming, we were able to refer to these names rather than the actual (x, y) coordinate. The following places were labelled as important:

Lander, Red Side Crater, Blue Side Crater, VuMarks, Field Corners, Sampling At Red Crater, Sampling At Blue Crater, Sampling At Red Depot, Sampling At Blue Depot, Red Depot, Blue Depot

Along with a positional (x, y) global map, we wanted to create an orientational global map to establish a consistent angle at any point on the map. The angle map was modeled off of the *Unit Circle*, which was used as a standard for marking angles on the map.

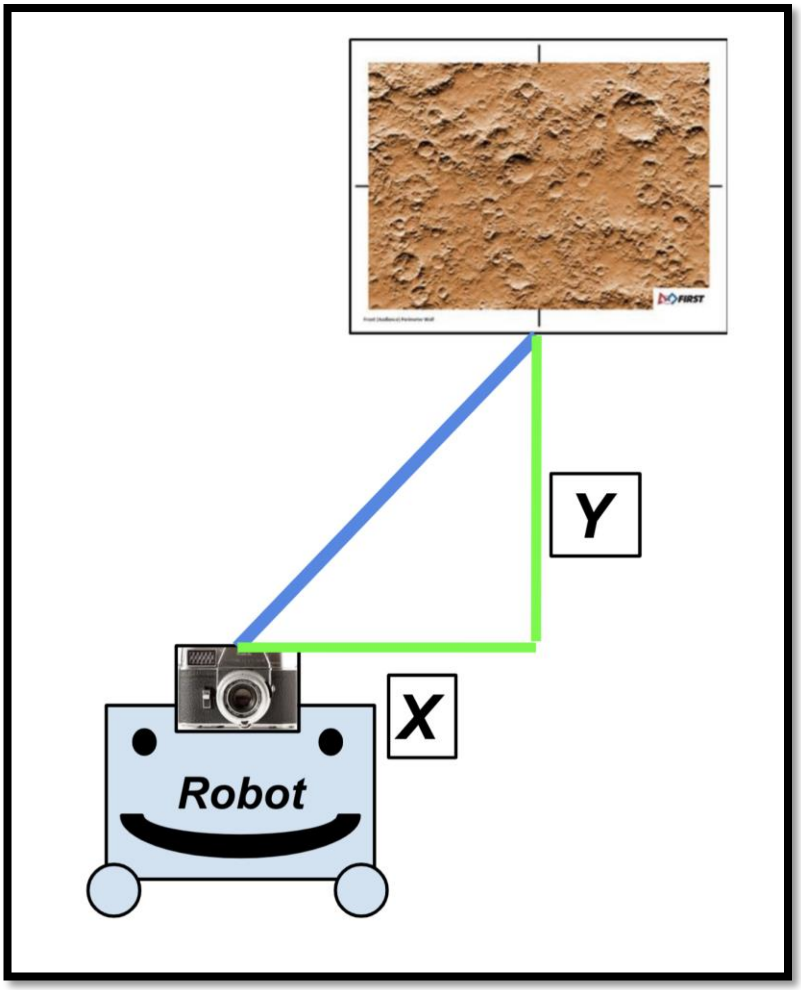

90°
0°
180°
270°

Localization

See engineering notebook entries: C42, C43, C146, C147, C148

In terms of our programming team, localization means to find out the robot's exact global (x, y) position on the field. In this case, the robot would have to find out its global (x, y) position on the **MOEPS global field grid** (check *Defining The Field Grid*). To accomplish this, we make use of the **Vuforia** image recognition technology and the **REV 2m Distance Sensor**.

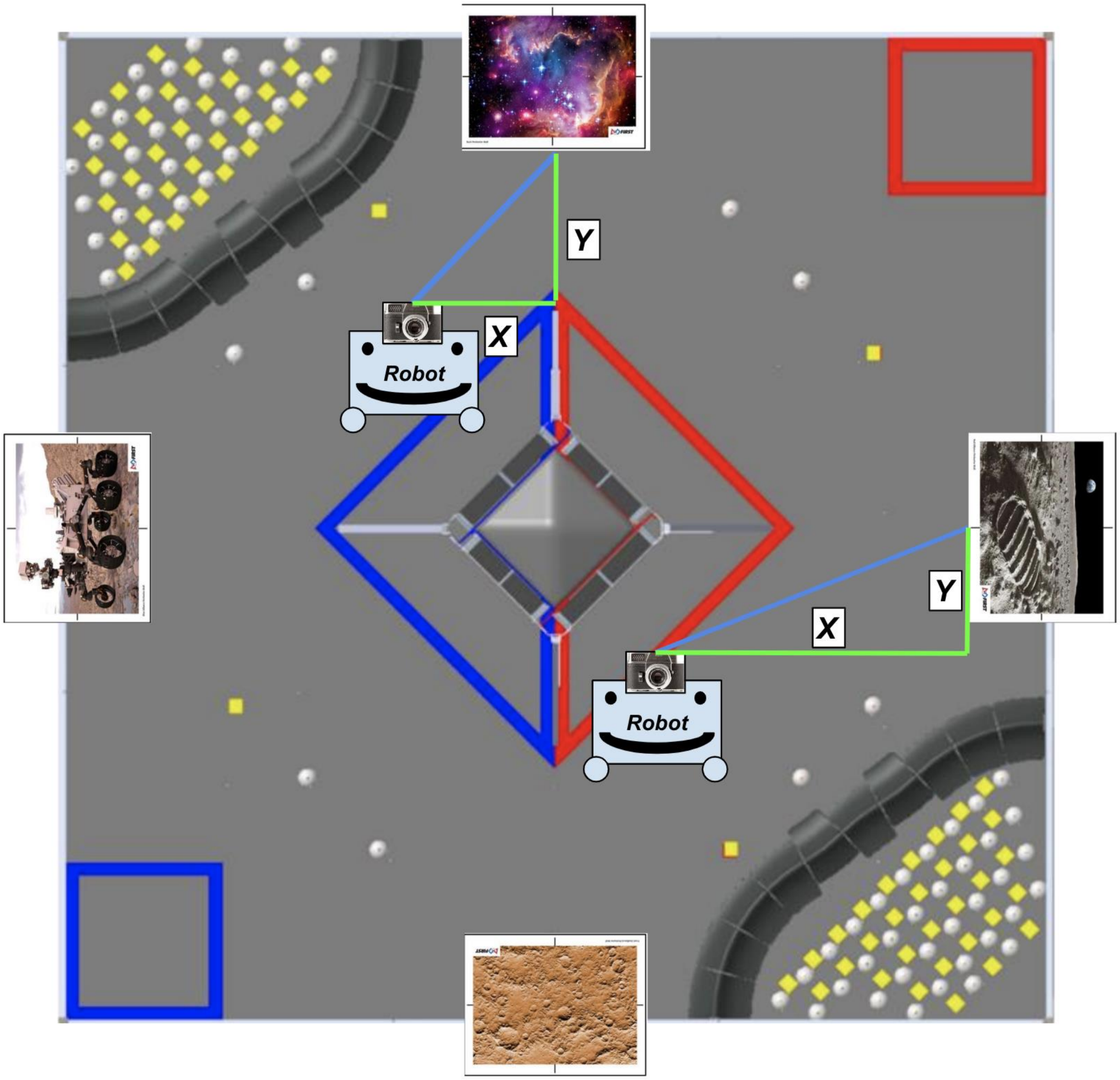
VuMark Localization



VuMark Localization on the field

When the robot's webcam sees a VuMark, the following steps are taken:

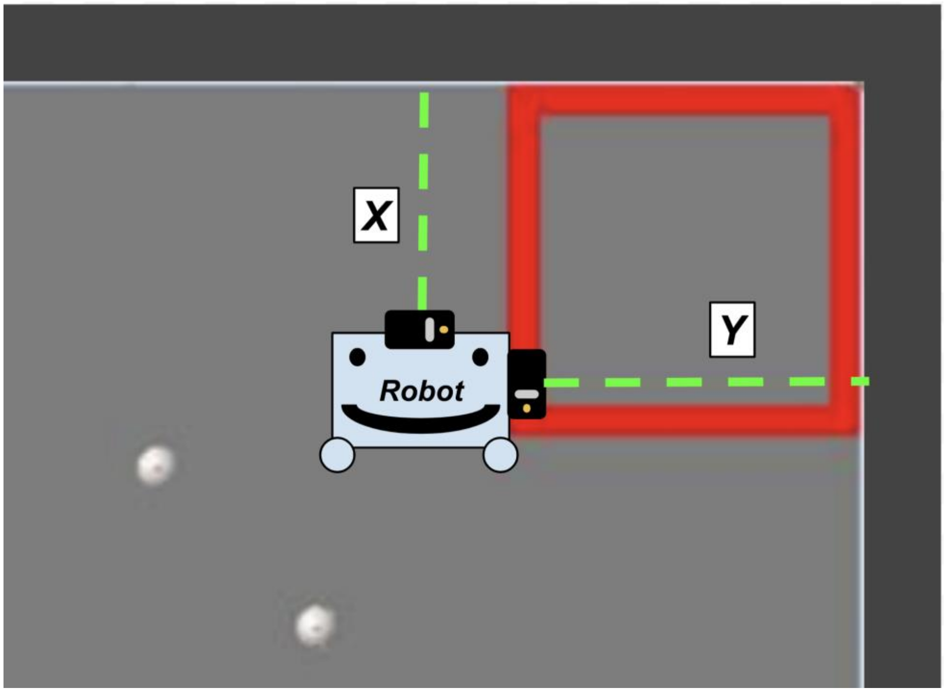
1. Extrapolate horizontal (x) and vertical (y) distance from the VuMark using Vuforia
2. Scale the x, y distance into our 2-inch units – this is done by multiplying the values by the scalar 1/50
3. Depending on the VuMark, subtract the x, y values as appropriate – each VuMark has a distinct x, y position on the field, so subtract the robot's local x, y position from the VuMark from the VuMark's global position



Distance Sensor Localization

Localization using the distance sensor is similar to the VuMark Localization method, but is less reliable due to inaccuracies and errors that occasionally occur in the distance sensors. It used when no VuMarks are available and localization is necessary.

In this case, the following steps are taken: 1. Get vertical and horizontal distance in inches from wall with distance sensors 2.



Subtract inches from wall (x, y) position

3. Resulting (x, y) coordinate is the robot's global location

Multithreading

Implementation of lift mechanism: C140

Implementation of realignment: C58, C59

Multithreading is a technique by which a single set of code can be used by several processors at different stages of execution. In other words, a program can have multiple sets of instructions running at the same time. With multithreading, the robot is able to do *more than one task* at any given time. Since our robot is trying to accomplish all standard autonomous points, time is often cut close to 30 seconds.

Without the use of multithreading, the robot's autonomous routine would have to speed up its motors significantly to meet the allotted 30 seconds. This speeding up results in less accuracy, resulting in an autonomous that is more prone to error.

Although processes like Vuforia and TensorFlow may run on separate threads, we intentionally use our own threads or pull from other threads for the following purposes:

- Bringing down the lift mechanism used for dropping/hanging
- Pathing algorithm realignment (see *Vuforia Listener* in *Additional Summary Information* for more details)

We also used **Atomic variables** for thread-safe operation. When a global variable is dealt with between 2 or more threads, there is always the danger of it leaking data when operations on it are done at the same time. Since using a raw variable without synchronization or any other standard is considered bad practice, we decided to use Atomic variables for thread-safe operation. This way, when communicating between the Main Robot thread and the Vuforia thread, we can guarantee that no strange behavior in autonomous occurs because of loss of data between the two threads.

Turning Methods

Conceptualization and implementation: C31, C32

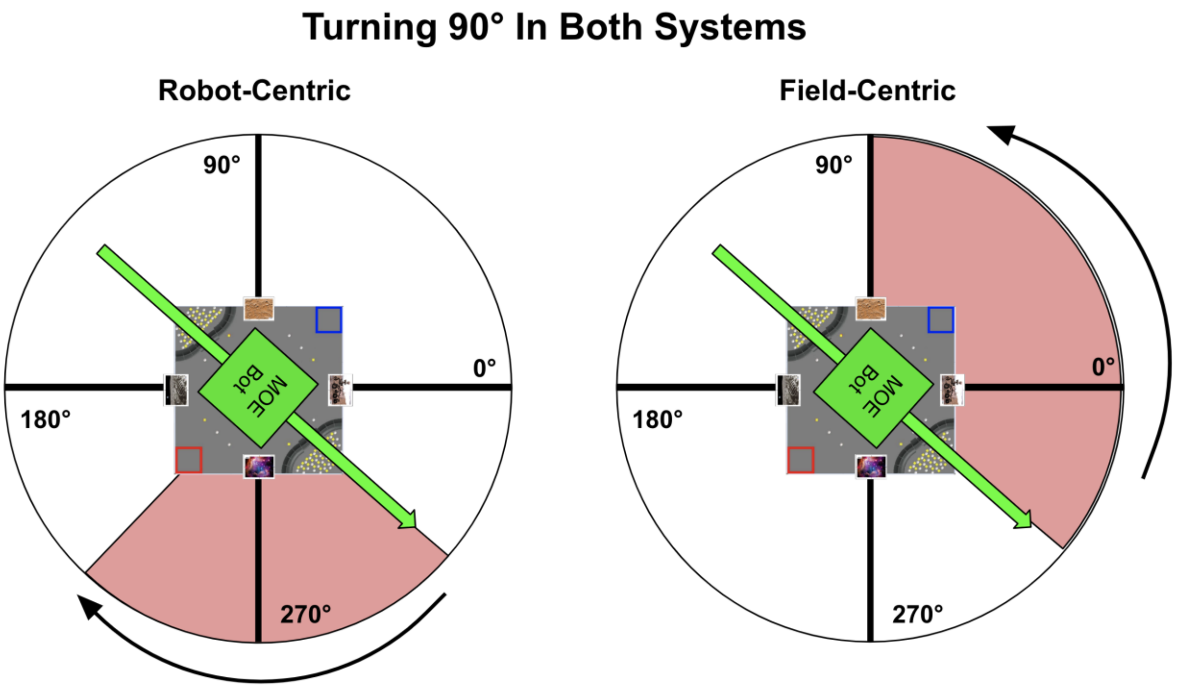
Turning in autonomous has to be precise to the degree for repeatable results, which is why turning is dictated by the **IMU** sensor built into the ***REV Expansion Hubs***. Instead of turning by time, we turn by setting the powers the motors and simply wait for the **IMU** to indicate that we are within the correct angle.

Field-Centric Turning & Robot-Centric Turning

In **field-centric turning,** the **MOEPS angle map** described above (see *Field Grid & MOEPS*), the robot turns to a given global angle on the field.

In **robot-centric turning** the robot turns to a given angle relative to itself.

Robot-centric vs. Field-centric Turning:



As shown in the diagram above, the robot turns 90° directly to the right in the robot-centric turn, while the robot is turning to the 90° mark in the field-centric turn. No matter the orientation, the robot will always turn to the same 90° mark in field-centric turning.

A* Pathfinding Algorithm & Dijkstra's Algorithm

*Conceptualization and implementation of old linear pathfinding algorithm (**not used on robot**):* C42, C43

Conceptualization and implementation: C46, C47

Implementation and testing: C50, C51

Debugging and gradual improvements: C55, C56, C58, C59, C64, C76, C93

Radius and size reductions: C97, C98, C99

2nd Stage Debugging: C104, C105

8-Directional Movement: C128, C129

3rd Stage Testing: C146

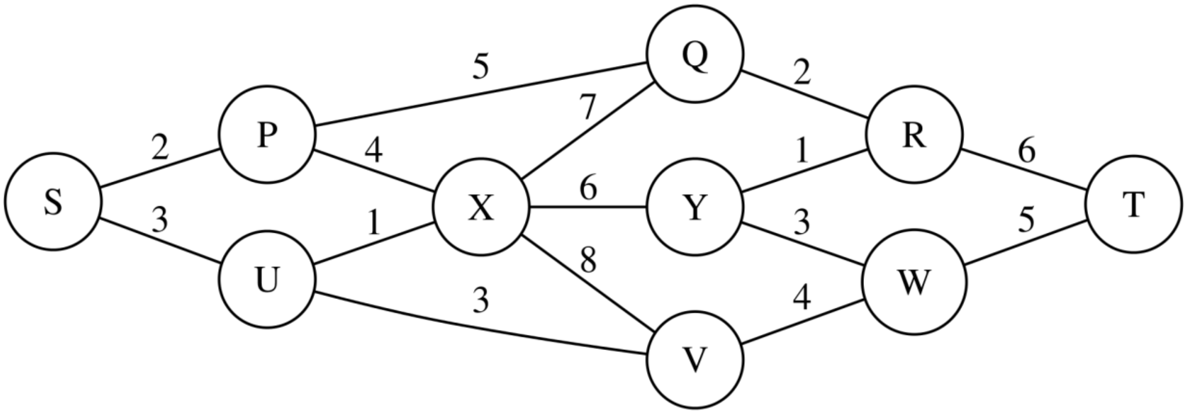
"Rotational Symmetry": C150, C151

Introduction

The purpose of the algorithm is to allow the robot to dynamically figure out how to reach its destination. In many autonomous pathings, whenever there is a slight disturbance, the autonomous fails to finish. Rather than explicitly giving the robot a path to follow, the robot is given an end destination. Through localization, the robot figures out its (x, y) coordinate on the field and calculates on its own how to reach the destination.

The *A* Pathfinding Algorithm (pronounced A Star)* is similar to the popular *Dijkstra's Algorithm*, which is used for finding the shortest paths between nodes in a graph. The only difference between the two is that the A* Algorithm utilizes a "heuristic function", or an approximation function, to approximate a faster solution to Dijkstra's algorithm. Dijkstra's algorithm checks many more cases than the A* Algorithm, therefore taking longer to arrive at a similar answer. Since the field we are using is 72x72 (5184) nodes, we wanted to guarantee that processing speed would be optimal in all situations and have two approaches available to use. The algorithms commonly deal with graphs shown like the one below, but had to be specially adapted in our case to work with a 2D grid.

Visual representation of traditional graph in computer science:



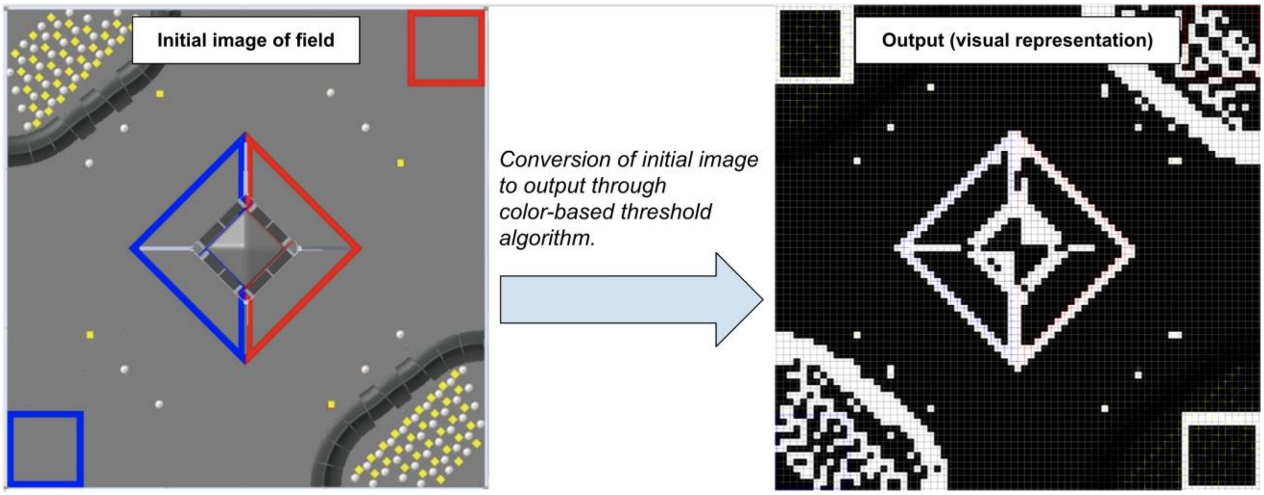
To account for processing speed & time, Dijkstra's Algorithm has a worse case time complexity (when using lists) of **O(N^2)** where N = number of nodes on the graph, while A* has a time complexity of **O(b^d)**, where b = branching factor and d = depth of the solution on the search tree. Note that the time complexity of A* is worse when using a very expensive heuristic cost function, but we are using the simple Euclidean distance, or the distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

With the pathfinding algorithms, the robot is able to move in **8 directions** to go from point A to point B. The directions are labelled as follows: **North, Northeast, East, Southeast, South, Southwest, West, and Northwest.**

Setup

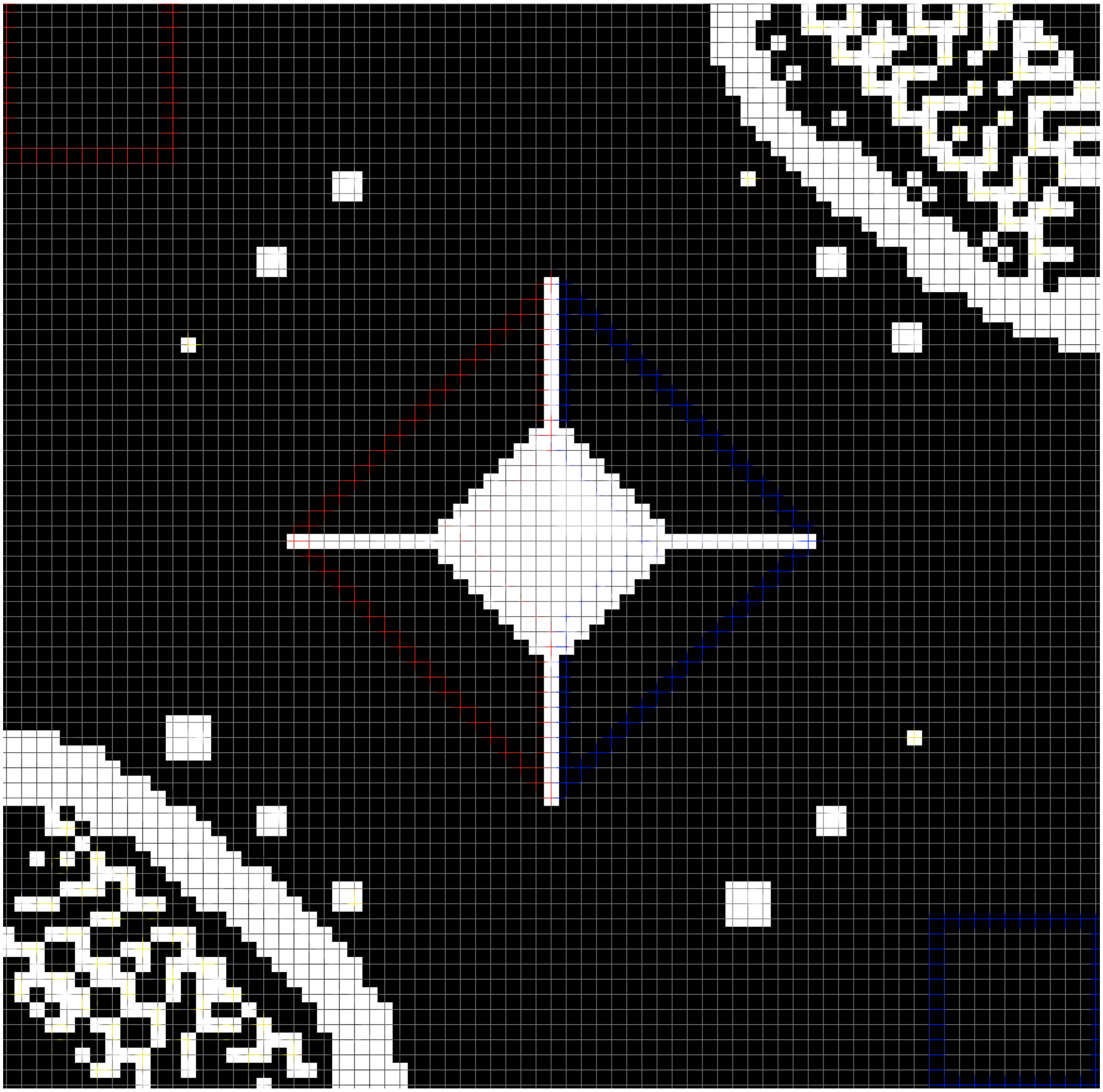
To utilize the **A* and Dijkstra algorithms**, we needed to first setup a graph. To accomplish this, we took a 2D image of the field from Game Manual 2. After that, we wrote a **Python** script (utilizing the *PIL imaging library*) to go through the image, converting it to points we deemed as barriers (white) and points we deemed as free space the robot could travel on (black). This conversion was done through a color-based threshold. In essence, the gray parts of the map were free space while the other colors were barriers. The output was an image with the converted points as well as a 72x72 *2-dimensional array* that we would be able to use as our graph for the A* algorithm. Also, the image was flipped because we wanted [0,0] of the 2D array to be the corner of the red depot, and [71,71] of the 2D array to be the corner of the blue depot.


Initial image of field
Conversion of initial image to output through color-based threshold algorithm.
Output (visual representation)

Mapped FTC Field (Visual Representation of Array)

1 (or white) = point a robot cannot travel on

0 (or black) = point a robot can travel on



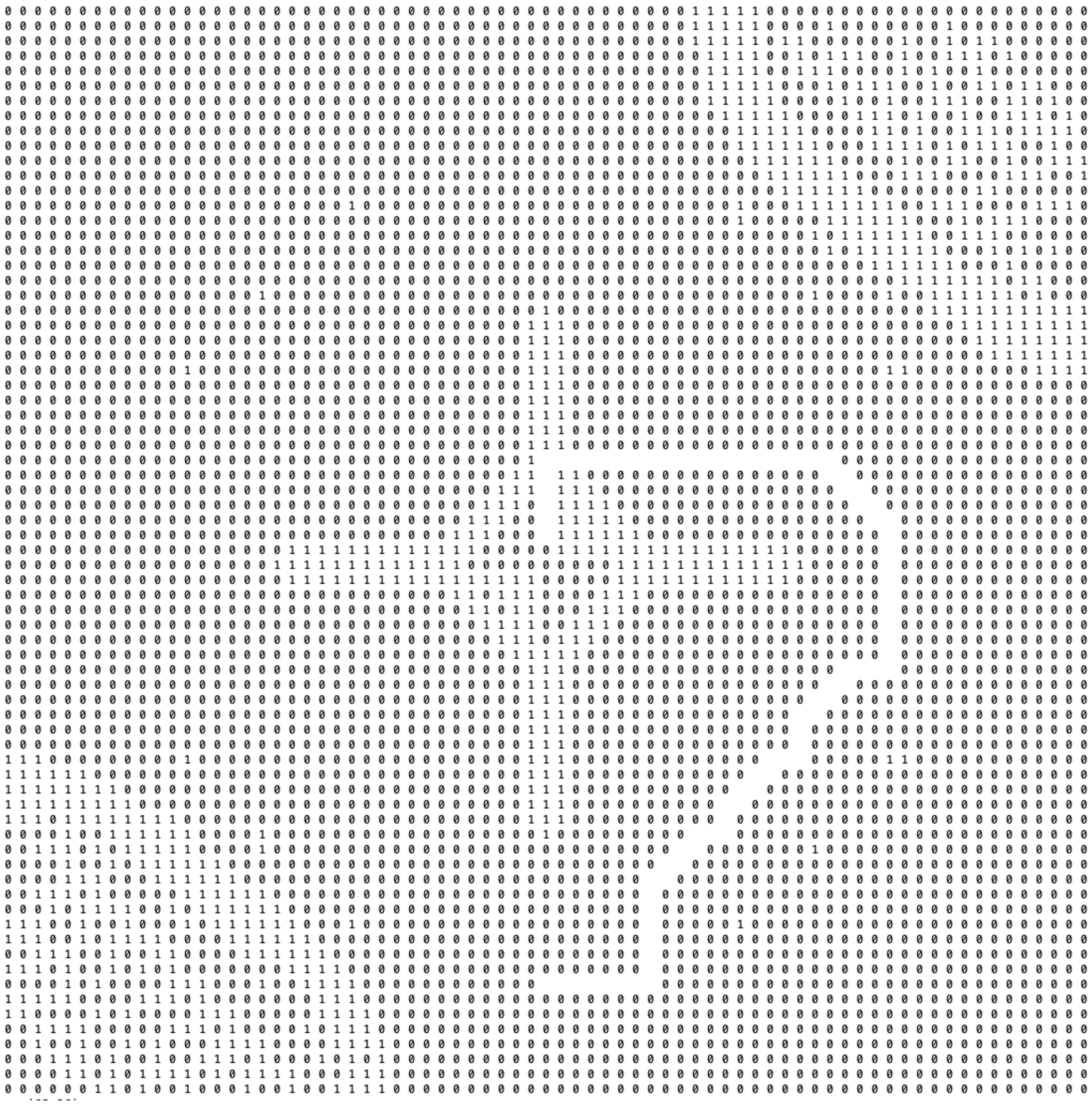
The original conversion had some errors, because places (depot, lines near the lander, etc...) were marked in white when they should have been open space. To fix this, we manually changed some values in the array. Since most of the conversion work was done by the Python script, this only took a few minutes.

The above image is what the output array looked like visually. The barriers shown in the image above are represented by 1s, while the free space shown in the image above are represented by 0s.

Implementation

We implemented the algorithm in Java, making a separate class to handle the calculations. To verify that we wrote the algorithm correctly and be able to make predictions on robot movements, we made a simulation to show the A* Algorithm's path from any Point A to Point B visually:

Initial Simulation

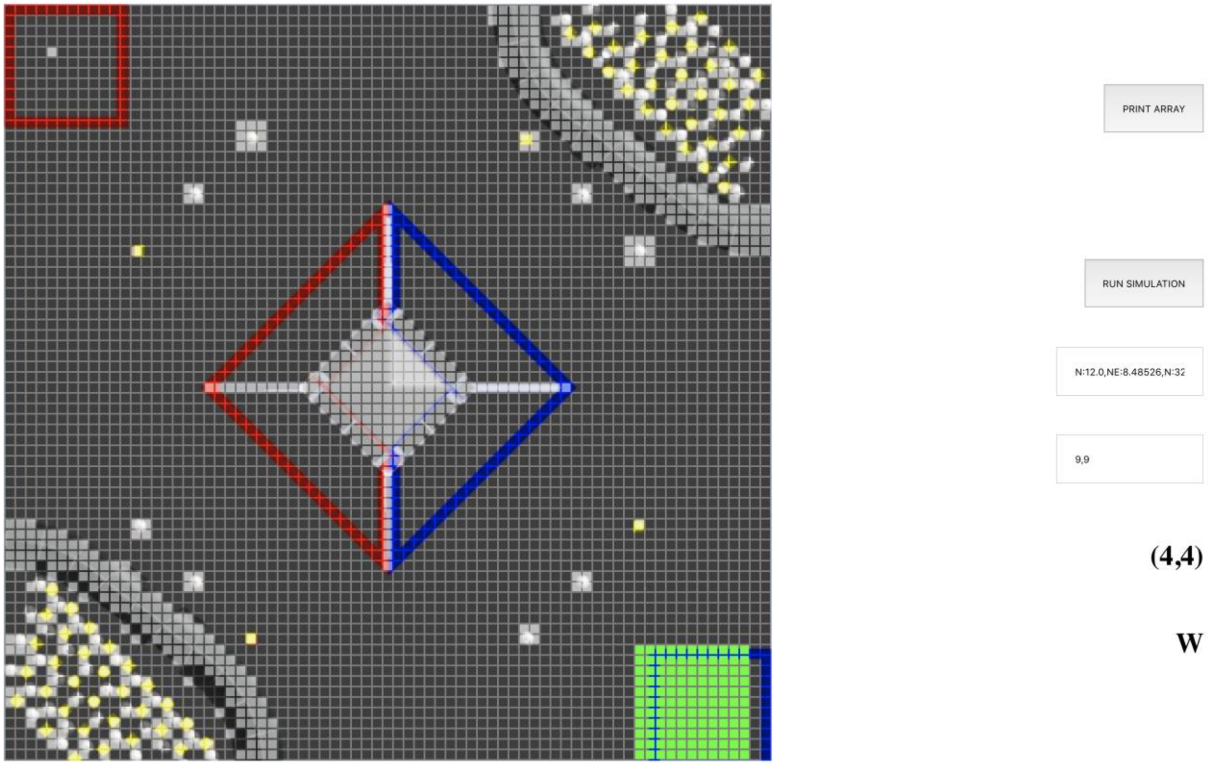


```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1                                         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1   1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1   1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0   1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0   1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0   1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0       0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0                 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

As complications with the algorithm increased, there was a need for a better simulation that more accurately depicted the algorithm in action. Below is a screenshot of an active simulation that shows the the robot (the green square), step by step, moving through the field.

Screenshot of Final Simulation

Video of live simulation: https://1drv.ms/v/s!AqOPfHs4_986ihlsFJLdiAYHtxG2



N:12.0,NE:8.48526,N:32

9,9

(4,4)

W

To now use the algorithm in practice, we had to convert the results into a usable format by writing an algorithm to do so.

Path Conversion Algorithm

(Input) - Original Pathfinding Results:	**(Output) - Usable Results:**
A series of points describing each point to go from point A to point B. *For example, getting from (0,0) to (5,5) could be:* (0,0) --> (0,1) --> (1,1) --> (2, 1) --> (3, 1) --> (3, 0) --> (4, 0) --> (4, 1) --> (5, 1) --> (5, 2) --> (5, 3) --> (5, 4) --> (5, 5)	The number of inches in each direction the robot has to go, in order (each unit is 2 inches). *For example, getting from (0,0) to (5,5) could be:* FORWARD 2 in. --> RIGHT 6 in. --> BACKWARD 2 in. --> RIGHT 2 in. --> FORWARD 2 in. --> RIGHT 2 in. --> FORWARD 8 in.

The robot first turns towards the **90° mark** described in the **MOEPS global angle map (**the direction facing the Crater/Mars VuMark). The results from the pathfinding algorithm would then be translated into movements for the robot based on encoder ticks. The result of this extensive process is a robust and repeatable movement system that allows the robot to figure out its own path when given two points on the field. This simplifies the process of adjusting and programming autonomous, as well as allowing for a more robust and dynamic movement system.

Pathfinding Algorithm Error Correction

The pathfinding algorithm worked perfectly well in theory, but in practice, there were a few issues we had to fix in order of importance.

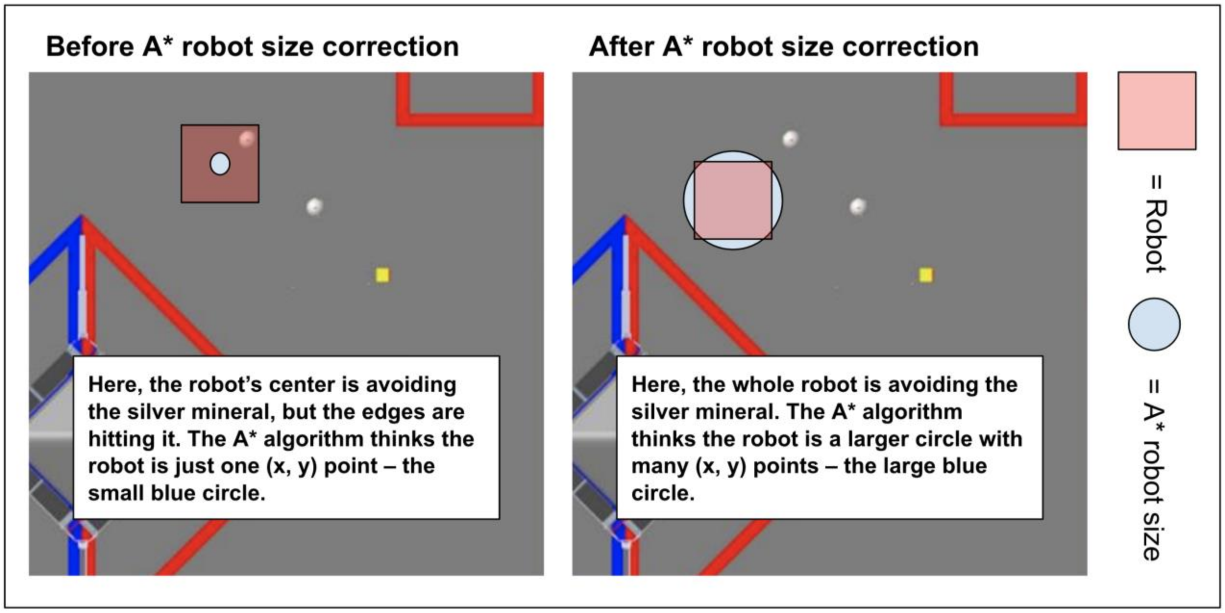
1. The **A* algorithm** treated the robot as a single (x, y) point on the **global 72x72 grid** while the robot actually comprised at least a circle of many points with radius about 10 inches. This led to the edges of the robot crashing into parts of the field (lander, crater, sampling, etc...) while its center thought it was following the pathfinding algorithms as a single small point.
2. While moving, the robot would turn slightly off angle. It would be not exactly at its end destination due to slight turning while making up, down, left, and right movements.

Error #1 – Size Corrections

Conceptualization and implementation: C55, C97

Second iteration: C98, C99

To fix this error, we modified the size of the robot in the algorithms. Instead of treating the robot as a single point, we treated it as a collection of multiple points – when put together, these points would form the robot rather than one small point.


Before A* robot size correction
Here, the robot's center is avoiding the silver mineral, but the edges are hitting it. The A* algorithm thinks the robot is just one (x, y) point – the small blue circle.
After A* robot size correction
Here, the whole robot is avoiding the silver mineral. The A* algorithm thinks the robot is a larger circle with many (x, y) points – the large blue circle.
= Robot
= A* robot size

Error #2 – Turn Corrections

To fix this error, we took the IMU sensor's horizontal angle before the robot followed the A* algorithm. We then constantly tracked the gyro sensor's angle while the robot followed the A* algorithm.

If the IMU's angle strayed by more than 2°, the robot self-corrected itself back to the correct angle by again utilizing the gyro to turn back into position.

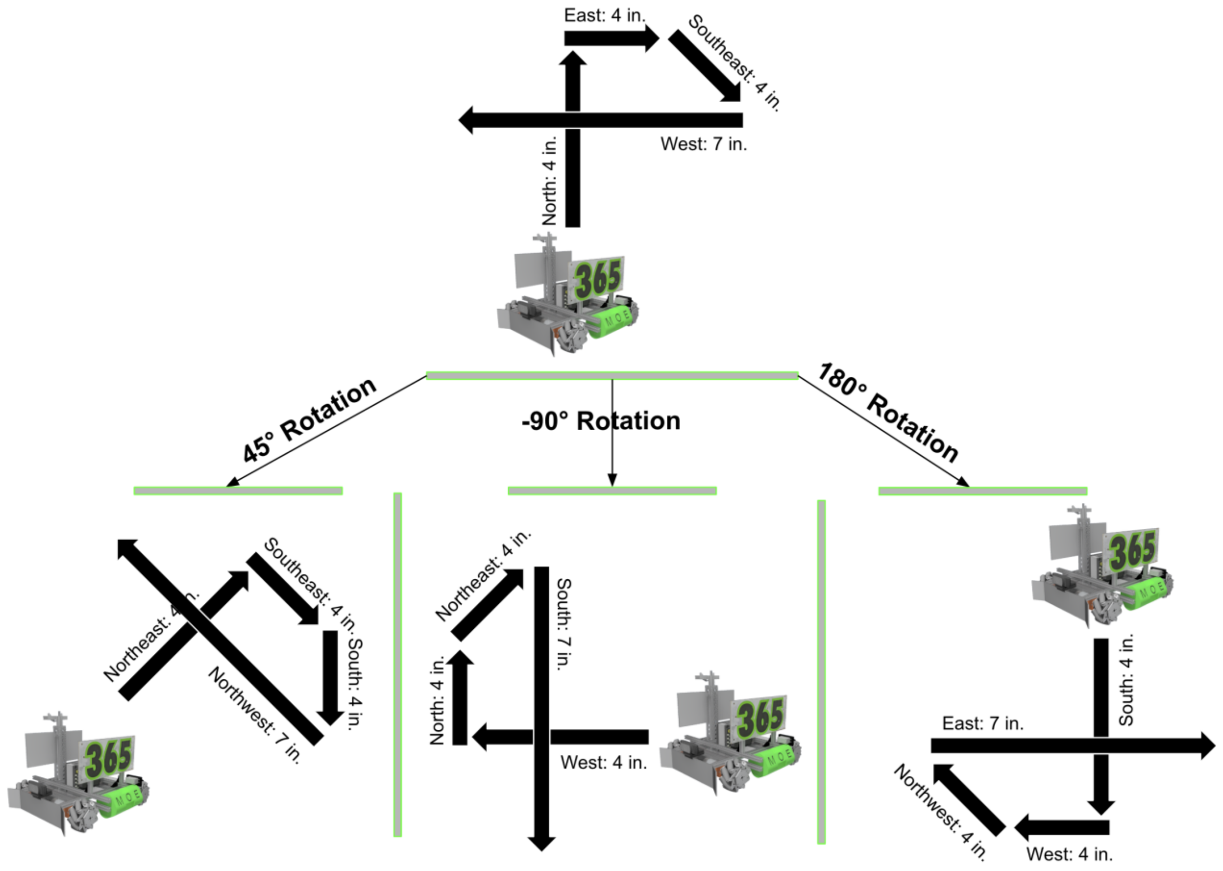
"Rotational Symmetry"

Conceptualization and Implementation: C150, C151

Another feature that we added to the pathfinding algorithm is the idea of "rotational symmetry". In other words, a given set of output instructions can be rotated by certain number of degrees while still preserving the relative directions of each movement.

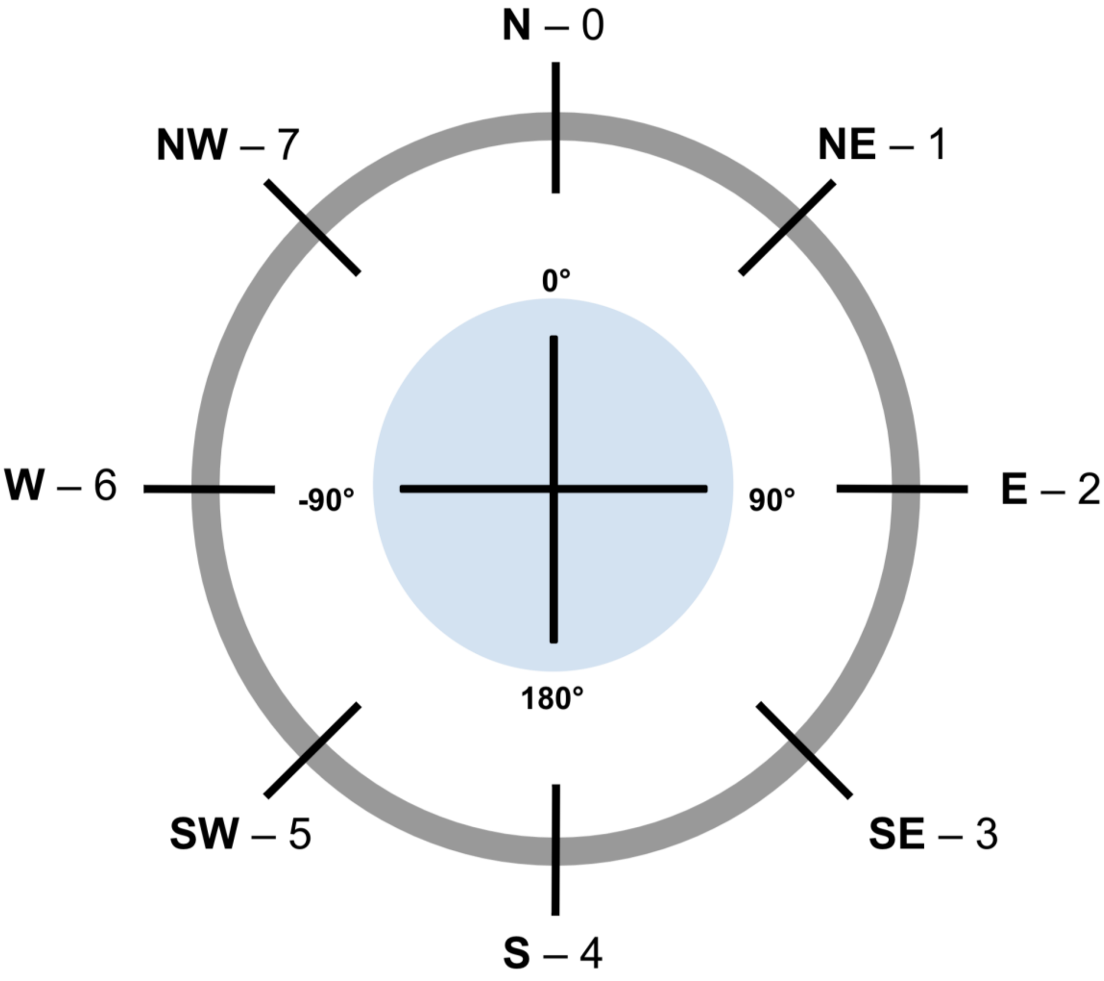
The purpose of rotational symmetry is to allow for optimizations in accuracy and speed of robot movements. Since moving forwards and backwards is always faster than strafing, rotational symmetry allows the robot to take a pathing from the Pathfinding Algorithms and rotate the pathing instructions. The robot can then quickly turn and apply the rotated instructions to allow for more forwards and backwards movements, resulting in a more robust movement.

Examples of rotations done on set of output instructions:



The algorithm is accomplished by setting a numerical value to each of the directions in clockwise order. North all the way around to Northwest is numbered from 0 up to 7. This simple numbering pattern makes rotation much simpler than writing each direction's rotation to its right. To rotate a direction, divide the angle needed to rotate by 45° and add it to the number. In the case a value goes above 7, it is wrapped back around to start at 0 (ie: 9 becomes 2).

Visualization of Direction to Number mapping, along with degrees associated with rotations:



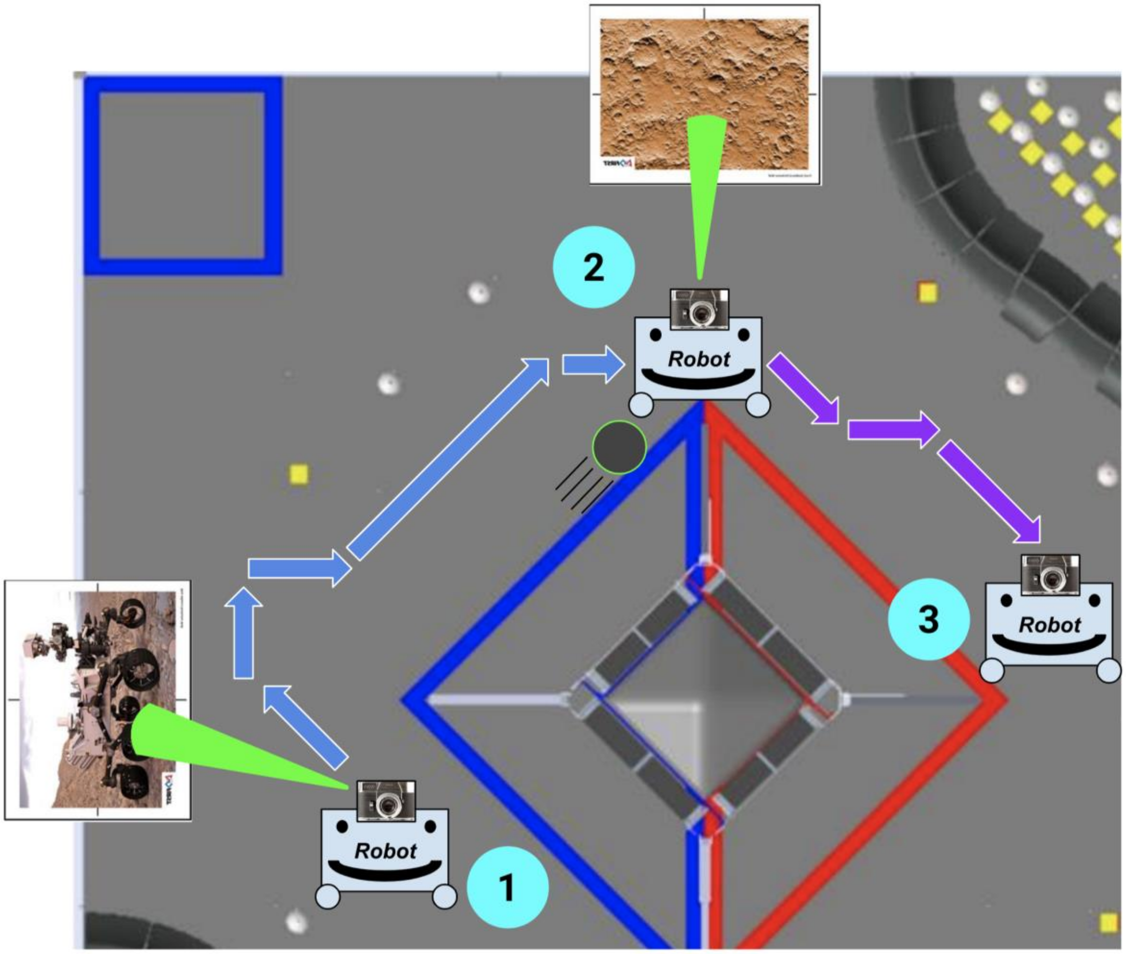
This system's simplicity becomes apparent when put into practice. For example, if an instruction says to move the robot North, the direction North's numerical value is 0. To rotate it by 90° clockwise, divide 90° by 45°, which equals 2. The 2 is added to North's numerical value, which results in 0+2 = 2. The 2 corresponds to the East direction, which is exactly a 90° clockwise rotation from North.

Realignment

Conceptualization and Implementation: C58, C59

As there is always a chance for error, such as another robot or debris in the way of a robot, a given robot might be knocked out of its planned path. This is another application for the Pathfinding Algorithms. During the course of following a path from the algorithm, the robot is always on the lookout for a new VuMark. (see *Vuforia Listener* in *Additional Summary Information*) If its camera sees one, it stops the current path it is on and restarts its pathing at the new VuMark. If the robot ever gets knocked off of its given path and fulfils the chance that it sees a new VuMark, the robot is able to get back on path.

The diagram below illustrates this process:



1. The robot localizes off of the Rover VuMark to figure out its (x, y) point
    a. Pathfinding Algorithms calculate a path to the destination (#3)
    b. Robot follows the pathing with encoders (blue arrows)
2. The robot is knocked off of its pathing by debris

    a. A new VuMark is seen and the robot stops its original pathing (blue arrows) and relocalizes, figuring out its new (x, y) point
    b. Pathfinding Algorithms calculate a path to the destination (#3)
    c. Robot follows the pathing with encoder (purple arrows)

3. Destination is reached

Error Correction & Fallback Plan B Routines

Throughout autonomous, there is opportunity for plenty of error to occur. Since a fundamental goal of autonomous is to have consistent, reproducible results, we try to better handle some errors that may result in a deviation from any planned autonomous route.

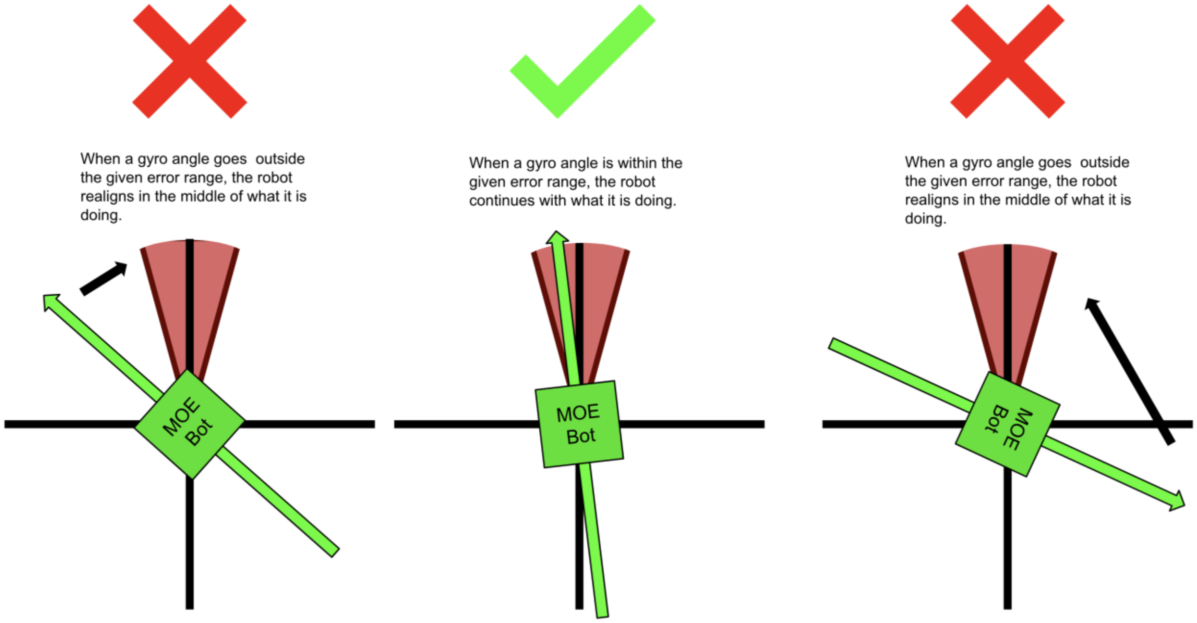
- Turn Corrections
- Distance Sensor Fallback

Turn Corrections

Implementation: C58, C59

In many of the routines and paths taken during autonomous, due to the nature of our mecanum wheels and the weight distribution of the robot, the robot gradually turns out of place. For example, when strafing normally for a period of 5 seconds, the robot could possibly turn 3° away from its starting angle. Over time, this error accumulates, and when sufficient, results in a faulty autonomous. To cut back on this, we define a given angle error range for any non-turning movement in autonomous. If the robot deviates from this error range, it interrupts what it is doing to turn back into proper position.

The diagram below shows the turn correction process.


When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.
MOE Bot
When a gyro angle is within the given error range, the robot continues with what it is doing.
MOE Bot
When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.
MOE Bot

Distance Sensor Fallback

Due the occasionally unreliable nature of the distance sensors, there is a need for a fallback when they produce errors. When getting readings in the middle of autonomous, a distance

sensor sometimes gives wildly out of range readings or errors. To fix this, we add a fallback. If the sensor does not give reasonable values within 2 seconds, the robot uses a preprogrammed point to plug into the Pathfinding Algorithms rather than a more accurate point from the distance sensors.

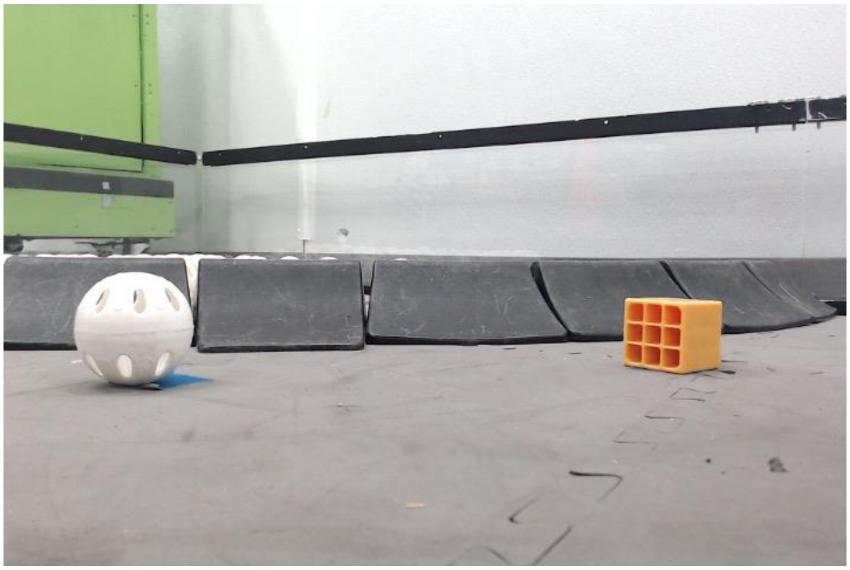
Sampling Algorithm

Our team tried using a variety of sampling methods, including the official TensorFlow neural network included in the SDK, creating our own neural network (see *Additional Summary Section*), and OpenCV for color detection. However, one key flaw with all of these methods was inconsistency. Even though the official TensorFlow network worked most of the time, we found that when placed on a competition field with a *yellow background* (like a gym floor), gold minerals were sometimes improperly recognized. This led to complications during competitions. Due to the flaws with the official TensorFlow network, our robot very often picked the wrong mineral to sample.

To solve this issue, we wrote our own **custom algorithm** for the sake of sampling.

Steps:

1. Take camera data from Webcam

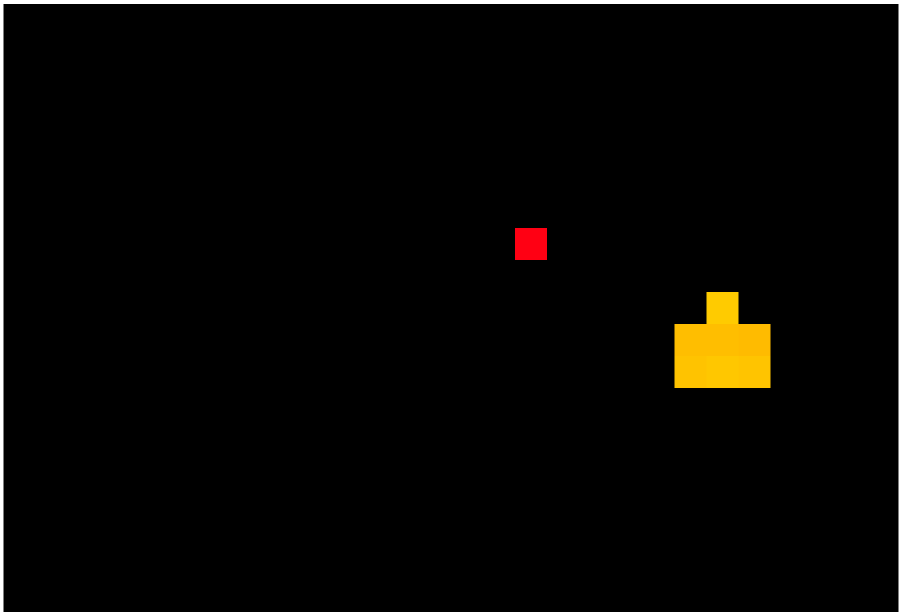


Example of camera data.

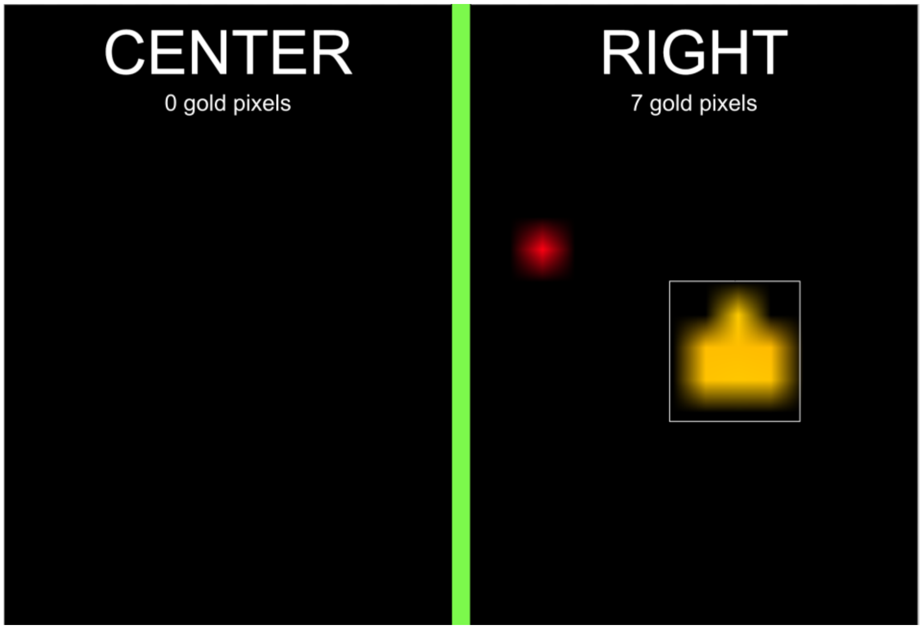
2. To remove any small splotches of yellow in the background (or any other strange color), reduce the resolution of the image.

Reduced resolution of original image.

3. Convert each pixel from RGB (Red, Green, Blue) color scheme to HSV (Hue, Saturation, Value) color scheme. Disregard any pixels without a high enough saturation value (< 0.5).



4. Compare amount of gold pixels in left and right side of image. The side that contains more gold pixels is considered the location of the gold sampling mineral. If both sides have less than 2 gold pixels, then the gold mineral is considered to be in the left location. (see *Gold Mineral Decision Algorithm* in *Additional Summary Section*)


CENTER
0 gold pixels
RIGHT
7 gold pixels

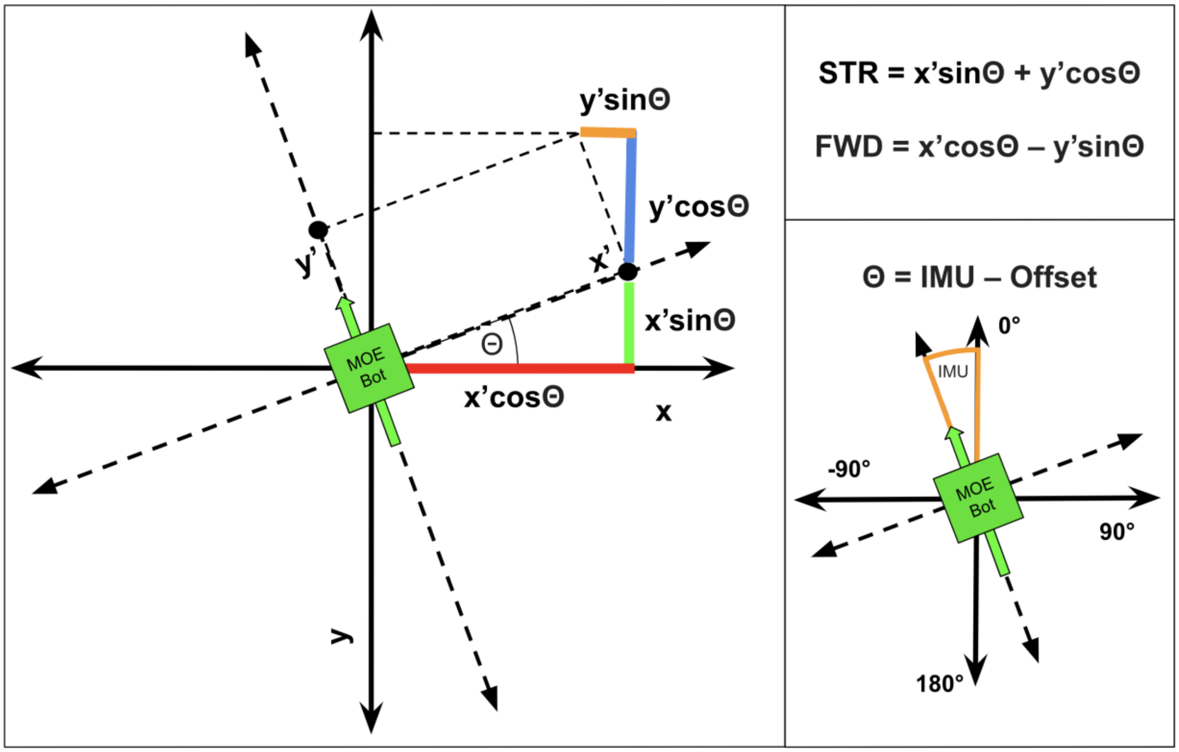
Driver Controlled Enhancements

Adjustable Field-Centric Movement

Conceptualization and implementation: C130, C131, C141

Since our robot uses a mecanum drive, it is capable of moving in any direction. Because of this, we are able to use **field-centric** motion rather than **robot-centric** motion. For the ease of the driver, we make all movements relative to the field rather than relative to the robot.

Note: In this diagram x' = Left Joystick X AND y' = Left Joystick Y



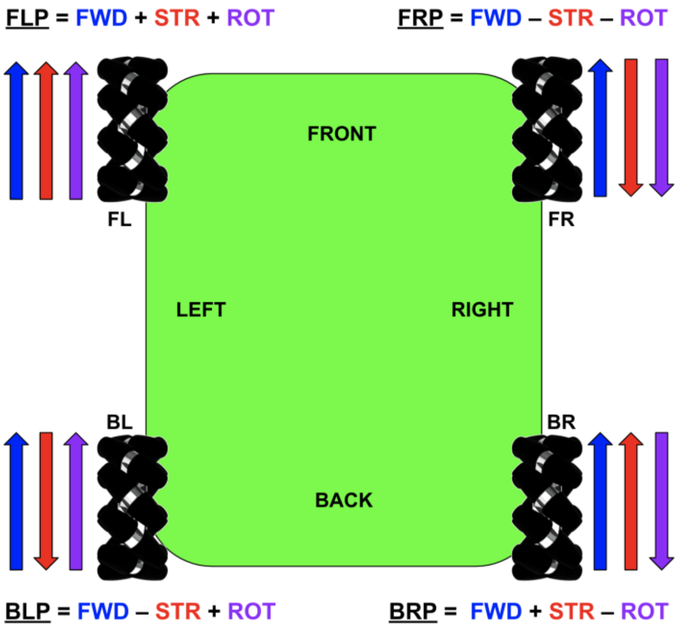
$$STR = x'\sin\Theta + y'\cos\Theta$$

$$FWD = x'\cos\Theta - y'\sin\Theta$$

$$\Theta = IMU - Offset$$

This diagram represents the **rotation of axes** that underlies the principles in field centric movement. The Θ is taken from the IMU, so that angle measurements are exact in the **field-centric** motion. Inputs for x' and y' are taken from the left joystick, and the right joystick x controls the rotation of the robot. Also, the offset for the IMU is to allow the driver to set a custom 0° point for the robot.

In the below diagram, the arrows indicate which direction the wheel turns when given a positive value for FWD (forward), STR (strafe), or ROT (rotation).



The corresponding values of FLP, FRP, BLP, and BRP are fed into the motors based on controller input so that field-centric movement occurs.

Lander Based Movement

Conceptualization and implementation: C141

Our robot also has "**lander specialized"** movement that allows for fine turning before hanging the robot. The D-pad allows for smaller adjustments in four-directional movement corresponding to the four buttons on the D-pad. Also, the bumpers allow for smaller adjustments in turning to fix orientation before raising the lift to hang.

Controls

Gamepad 1

Left Joystick: Field-centric movement in all directions

Right Joystick: Rotational movement (turns)

A: Reset 0° point (forward heading) for field-centric movement

Left Bumper: Turn slowly left (see *Lander Based Movement*)

Right Bumper: Turn slowly right (see *Lander Based Movement*)

D-Pad: (see *Lander Based Movement*)

UP → Move slowly forward

DOWN → Move slowly backward

LEFT → Strafe slowly left

RIGHT → Strafe slowly right

Autonomous Routines

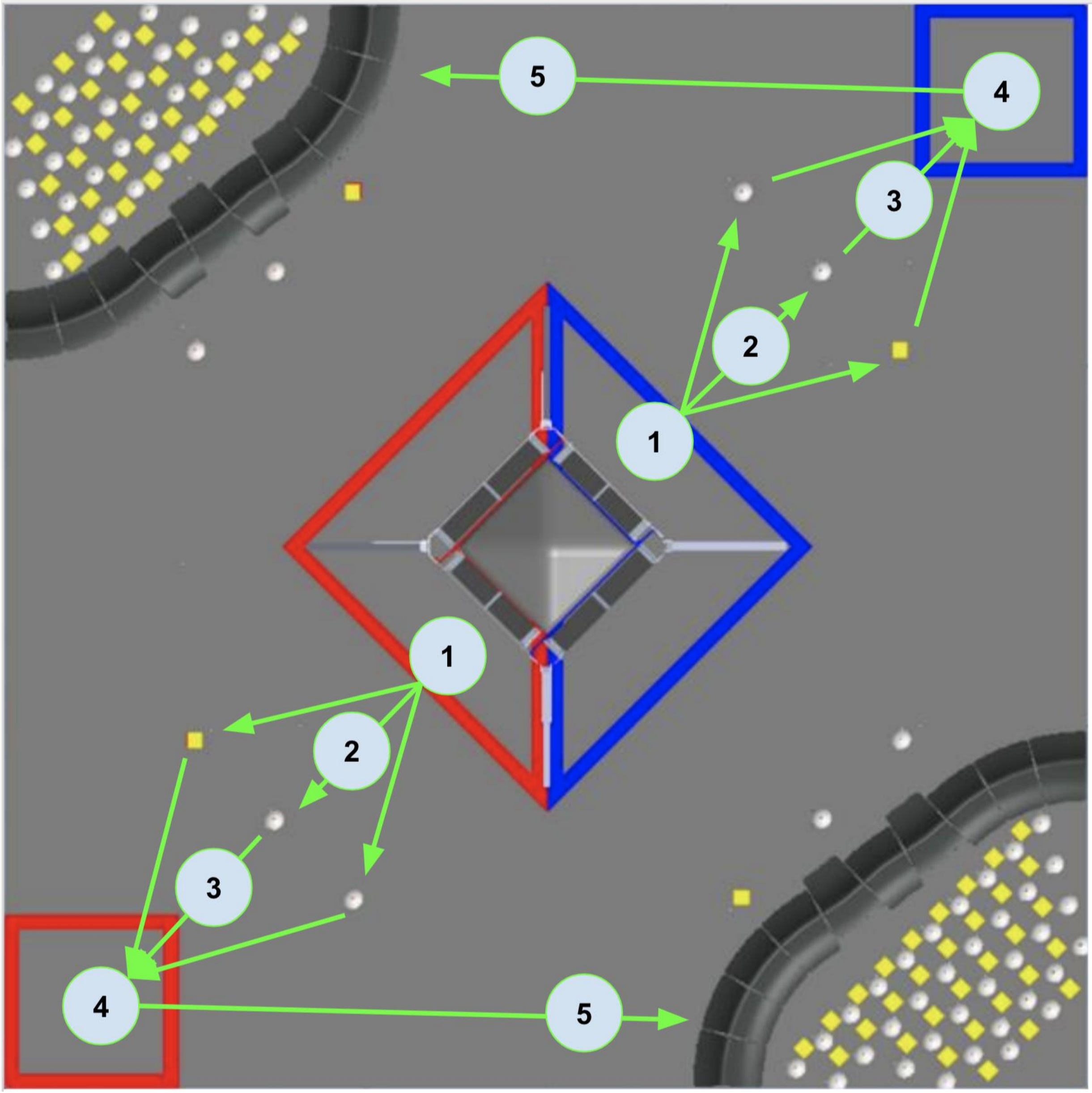
Conceptualization and testing of routine: C76

Iterative improvements, testing, and debugging: C84, C85

Testing environment fabrication: C104

2^{nd} *Stage Iterative improvements, testing, and debugging:* C105, C106, C121, C122

Testing and evaluations: C140, C141, C145



The image above represents our autonomous routine for the **depot starting points**. For the **crater starting points**, we only sample the gold mineral. We plan on executing the following steps for the depot starting points (the crater starting points are only the first two steps):

1. Landing & Detecting gold mineral
2. Travelling to and knocking off gold mineral
3. Travelling to depot
4. Deposit the Team Marker and leave gold mineral
5. Travel to crater & extend arm into crater

Initialization

1. Initialize motors, servos, sensors, and all other devices from the Hardware Map.
2. Initialize REV IMU sensor
3. Initialize Vuforia
4. Initialize TensorFlow
5. Reset Team Marker Servo to position 1
6. Reset Crater Extension Arm to position 1

1. Landing & Detecting gold mineral (30 pts.)

1. Use linear actuator to move lift up for given # of encoder tics
    a. The robot touches the floor and the claw goes above the top of the handle on the lander
2. Using multithreading, lower the lift back to its starting position
    a. (see *Multithreading* in *Key Algorithms*)
    b. The program continues on without waiting for the process to finish because of the multithreading
3. Turn ~70° to see 2 minerals and decide which is gold
    a. (see *Sampling Algorithm* in *Key Algorithms*)
4. The robot turns the appropriate # of degrees to face the gold mineral in left, right, or center

2. Landing & Detecting gold mineral (25 pts.)

1. Move forward appropriate # of inches to knock off gold mineral from its starting position
2. Continue moving forward to safely clear other 2 silver minerals

3. Travelling to depot

1. Turn appropriate number of degrees to face the depot
2. Move forward appropriate # of inches to reach the depot

   a. Note that the gold mineral is kept in control of the robot through the metal plate on its front – similar to a bulldozer

4. Deposit the Team Marker and leave gold mineral (17 pts.)

1. Turn appropriate # of degrees for front of robot to face the front/back wall
2. Localize and figure out (x, y) posi8tion on the MOEPS global field grid by using distance sensors
   a. (See *Distance Sensor Localization* in *Localization*)
   b. If the distance sensors have an error in measurement, fallback to Plan B
3. Turn appropriate number of degrees for left side of robot to face corner of field
4. Drop off Team Marker by setting Team Marker servo to position 0
5. Calculate path to crater using Pathfinding Algorithms
   a. ***Plan A:*** Use (x, y) position from distance sensors
   b. ***Plan B:*** Use default (x, y) position as an estimate – less accurate than Plan A
   c. (see *A* Pathfinding Algorithm & Dijkstra's Algorithm*)

5. Travel to crater & extend arm into crater (10 pts.)

1. Turn ~90° right for the back to face the crater
2. Apply 90° rotation to pathfinding algorithm results to fit new robot orientation
   a. (see *"Rotational Symmetry" in A* Pathfinding Algorithm & Dijkstra's Algorithm*)
3. Follow pathing to reach crater
4. Extend arm
5. Drive backwards to guarantee arm is in crater

Additional Summary Information

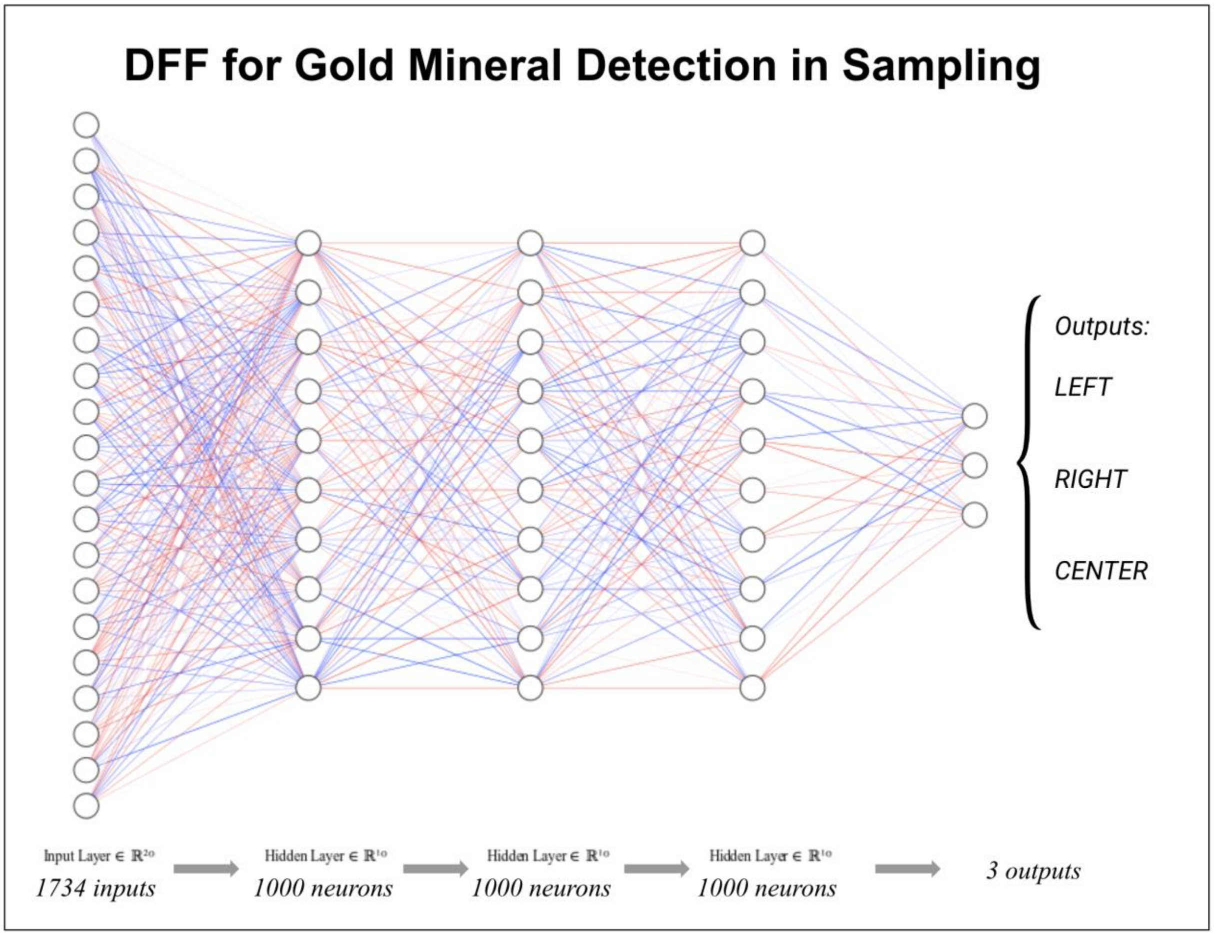
Creating An Artificial Neural Network (ANN)

Conceptualization and implementation: C110, C111, C112, C113, C114, C115

Before the integration of TensorFlow Lite into the official FTC App, we created a neural network with TensorFlow that could distinguish the left, center, or right position of the gold in the sampling minerals.

Choosing the Correct Structure

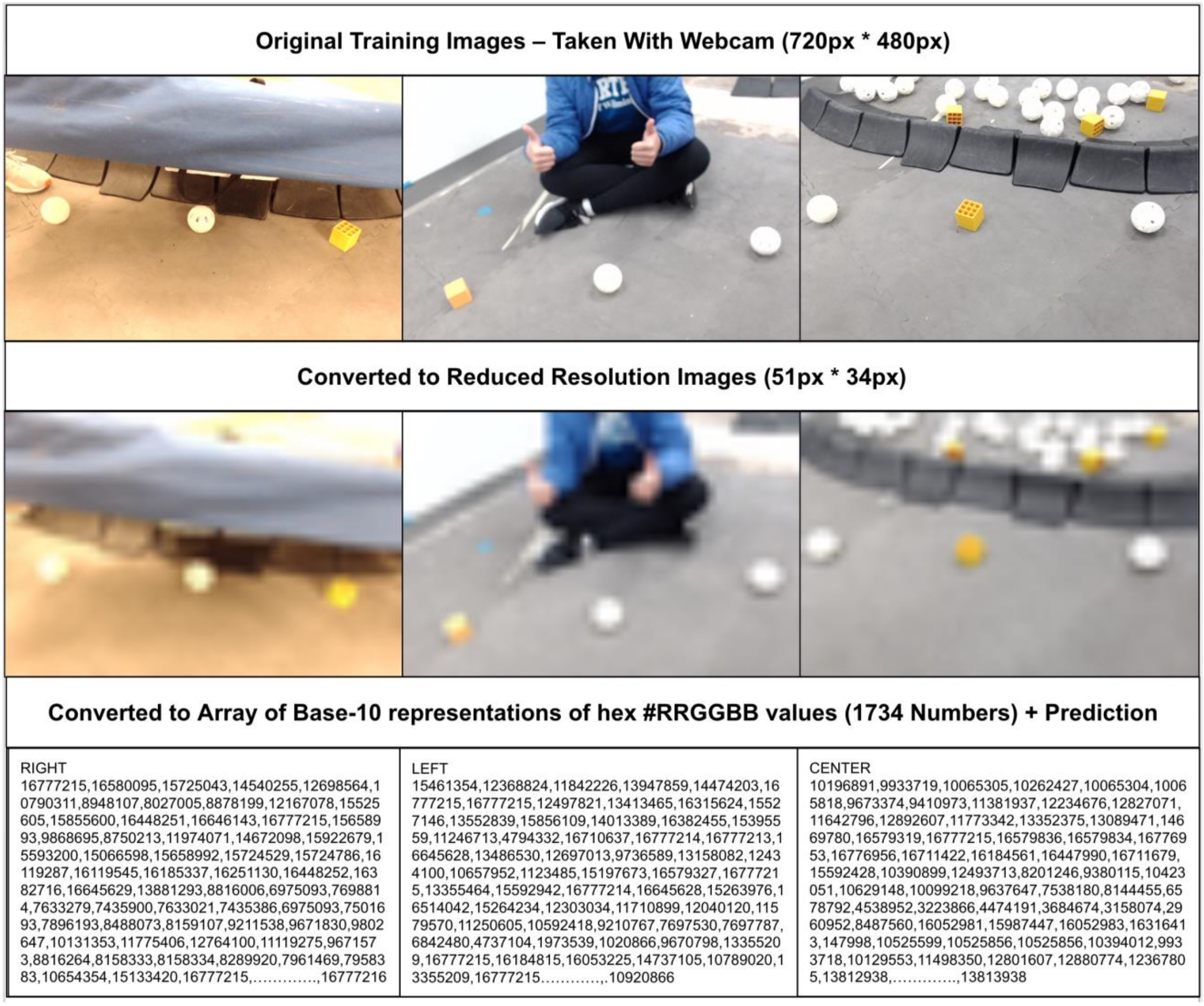
We decided to go with a ***deep feed-forward (DFF) neural network*** with ***backpropagation,*** a commonly used technique to train a neural network based around gradient descent.


DFF for Gold Mineral Detection in Sampling
Outputs:
LEFT
RIGHT
CENTER
Input Layer ∈ $\mathbb{R}^{20}$
1734 inputs
Hidden Layer ∈ $\mathbb{R}^{10}$
1000 neurons
Hidden Layer ∈ $\mathbb{R}^{10}$
1000 neurons
Hidden Layer ∈ $\mathbb{R}^{10}$
1000 neurons
3 outputs

There are 1734 inputs from the 1734 pixels in each of our input images, and 1000 neurons in each hidden layer. The final output has 3 possibilities. This neural network requires less training data because the problem at hand is fundamentally clear in terms of processing; there are no

complex edge detections required. All the neural network has to do is distinguish between yellow and white and their locations in the images.

Acquiring Training Data & Preprocessing (Total of 48 Images)



Original Training Images – Taken With Webcam (720px * 480px)

Converted to Reduced Resolution Images (51px * 34px)

Converted to Array of Base-10 representations of hex #RRGGBB values (1734 Numbers) + Prediction

RIGHT	LEFT	CENTER
16777215,16580095,15725043,14540255,12698564,10790311,8948107,8027005,8878199,12167078,15525605,15855600,16448251,16646143,16777215,15658993,9868695,8750213,11974071,14672098,15922679,15593200,15066598,15658992,15724529,15724786,16119287,16119545,16185337,16251130,16448252,16382716,16645629,13881293,8816006,6975093,7698814,7633279,7435900,7633021,7435386,6975093,7501693,7896193,8488073,8159107,9211538,9671830,9802647,10131353,11775406,12764100,11119275,9671573,8816264,8158333,8158334,8289920,7961469,7958383,10654354,15133420,16777215,............,16777216	15461354,12368824,11842226,13947859,14474203,16777215,16777215,12497821,13413465,16315624,15527146,13552839,15856109,14013389,16382455,15395559,11246713,4794332,16710637,16777214,16777213,16645628,13486530,12697013,9736589,13158082,12434100,10657952,1123485,15197673,16579327,16777215,13355464,15592942,16777214,16645628,15263976,16514042,15264234,12303034,11710899,12040120,11579570,11250605,10592418,9210767,7697530,7697787,6842480,4737104,1973539,1020866,9670798,13355209,16777215,16184815,16053225,14737105,10789020,13355209,16777215............,10920866	10196891,9933719,10065305,10262427,10065304,10065818,9673374,9410973,11381937,12234676,12827071,11642796,12892607,11773342,13352375,13089471,14669780,16579319,16777215,16579836,16579834,16776953,16776956,16711422,16184561,16447990,16711679,15592428,10390899,12493713,8201246,9380115,10423051,10629148,10099218,9637647,7538180,8144455,6578792,4538952,3223866,4474191,3684674,3158074,2960952,8487560,16052981,15987447,16052983,16316413,147998,10525599,10525856,10525856,10394012,9933718,10129553,11498350,12801607,12880774,12367805,13812938,............,13813938

Key ideas in images for the neural network:

- Changed background of images to show background does not matter
- Changed lighting of images to show lighting does not matter
- Changed tilt of images to show tilt does not matter
- Did not change order of minerals to show **only** ordering of minerals matter

NOTE: The preprocessing was automated using Python scripts to save time.

The reason the images' resolutions were reduced was due to the fact that training the network would require more time. Although training the network at full resolution would be fine, it

would possibly take a few minutes, and this can get cumbersome when refining and tweaking the data. We felt predictions could be made just as well at reduced resolution.

The process of converting to a Base-10 representation of hex #RRGGBB values:

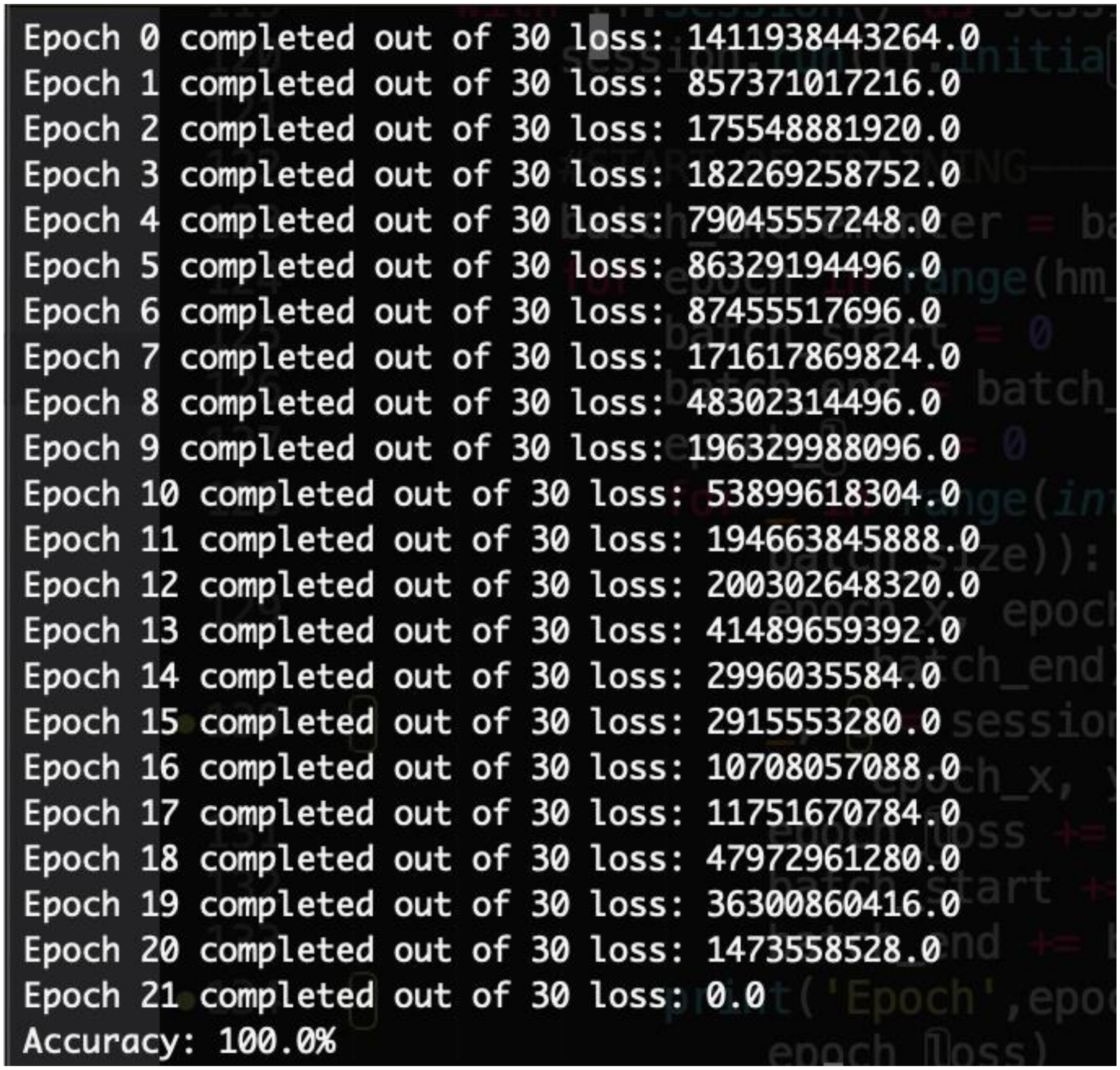
1. Take individual R, G, B (0 – 255) values of each pixel
2. Convert R, G, B values into one hexadecimal (base-16) number (#RRGGBB)
3. Take hexadecimal number #RRGGBB into a decimal (base-10) number

All the data was saved to a .txt file to be trained on later.

Training & Accuracy of Neural Network

Because we reduced the resolutions of the images in the preprocessing, training time for the 48 images was incredibly short: 5-15 seconds.

After training a successful model, here were our results.



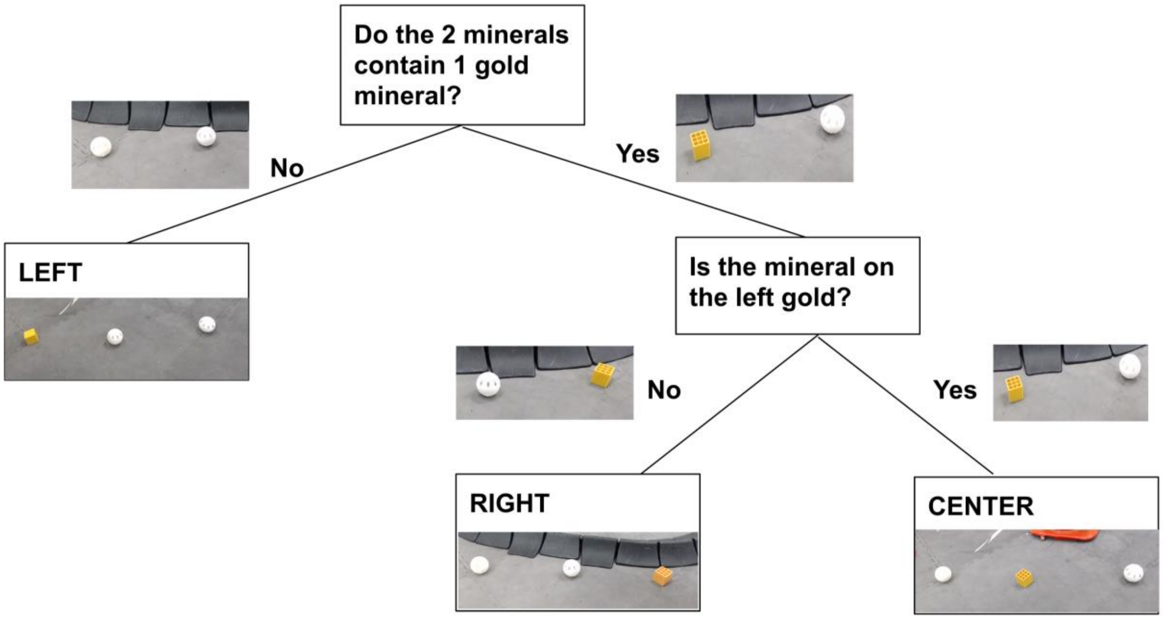
```
Epoch 0 completed out of 30 loss: 1411938443264.0
Epoch 1 completed out of 30 loss: 857371017216.0
Epoch 2 completed out of 30 loss: 175548881920.0
Epoch 3 completed out of 30 loss: 182269258752.0
Epoch 4 completed out of 30 loss: 79045557248.0
Epoch 5 completed out of 30 loss: 86329194496.0
Epoch 6 completed out of 30 loss: 87455517696.0
Epoch 7 completed out of 30 loss: 171617869824.0
Epoch 8 completed out of 30 loss: 48302314496.0
Epoch 9 completed out of 30 loss: 196329988096.0
Epoch 10 completed out of 30 loss: 53899618304.0
Epoch 11 completed out of 30 loss: 194663845888.0
Epoch 12 completed out of 30 loss: 200302648320.0
Epoch 13 completed out of 30 loss: 41489659392.0
Epoch 14 completed out of 30 loss: 2996035584.0
Epoch 15 completed out of 30 loss: 2915553280.0
Epoch 16 completed out of 30 loss: 10708057088.0
Epoch 17 completed out of 30 loss: 11751670784.0
Epoch 18 completed out of 30 loss: 47972961280.0
Epoch 19 completed out of 30 loss: 36300860416.0
Epoch 20 completed out of 30 loss: 1473558528.0
Epoch 21 completed out of 30 loss: 0.0
Accuracy: 100.0%
```

As shown on the image, the accuracy is 100% on our 48 images. The loss of 0.0 might be an indication of over-fitting the training data, but the network was able to successfully predict our test data, so the network indicates it has not overfitted to the point of inaccuracy. In other words, when we gave the network new data, it was able to successfully determine whether the gold mineral was in the left, right, or center.

Gold Mineral Decision Algorithm

See engineering notebook entries: C121, C122, C128, C129, C130

Since our robot was only able to see two minerals on the field, we had to write an algorithm to figure out which one is gold. To accomplish this, we guaranteed that the two minerals we saw would be the 2 right minerals out of the 3 in sampling. If the 2 were silver, the gold would be on the left. If the gold was the left of the 2 right minerals, it would be on the center. If the gold was on the right of the 2 right minerals, it would be on the right.


Do the 2 minerals contain 1 gold mineral?
No
Yes
LEFT
Is the mineral on the left gold?
No
Yes
RIGHT
CENTER

Vuforia Listener

(For the purpose of this, see *Realignment* in *A* Pathfinding Algorithm & Dijkstra's Algorithm*.)

To be able to retrieve the exact moment a Vuforia tag is seen by the camera, we took a unique approach in capturing these events. In Vuforia, there is a class known as the **VuforiaTrackableDefaultListener**. Normally, this class is called to check and retrieve a VuMark when one knows exactly when they will see a VuMark. However, due to the variable nature of our implementation of the A* and Dijkstra Pathfinding Algorithms (see *A* Pathfinding Algorithm & Dijkstra's Algorithm*), there is a need to know when a VuMark is found in a safe and efficient way.

This approach creates a class called **MOEListener** that extends the **VuforiaTrackableDefault Listener,** of which is used instead of the **VuforiaTrackableDefaultListener**. When overriding the methods in the default listener, we realized that there was no convenient method to realize when a *new* VuMark was found, so we modified one of the existing methods to let us know when a new VuMark was found. This way, in the event of a new VuMark, our robot would be notified properly.

Since Vuforia runs on a separate thread, there needed to be way to guarantee that the program would work consistently on multiple threads while being able to share information between the two. For this purpose, we utilized **Atomic** variables in Java for thread-safe sharing of information between the threads. If we did not, there is the slight chance that when two threads modify the same variable at the same time, there could be a loss of information. The usage of **Atomic** variables notifies the robot in the middle of following the path found by the Pathfinding Algorithms to stop what it is doing and realign against the VuMark.

Text-To-Speech (TTS)

For additional fun and utility, we incorporated the *Google Text-to-Speech* technology that allows text to be read aloud in a human-like fashion.

On the field, our robot likes to let us know how it is doing through a variety of phrases, including when it is initialized. Certain phrases include:

- “Initialized Vuforia”
- “Initialization Complete”
- “Initialized Gyro”

Over time, hearing these phrases can become cumbersome; however, our robot likes to spice things up. When completing certain tasks in autonomous, the robot likes to show its patriotism and loyalty to our team. Certain phrases it uses include:

- “Go MOE”
- “Whooo!”
- “Hi _____” (where _____ may be someone’s name)
    - Note: this is pre-programmed, we have not yet integrated the facial recognition technology for the robot to detect people on its own

Our robot also has a fondness for music, which it may play on or off the field. More than anything else, our robot’s personality makes interacting with it more interesting and fun! :)

MOE, Miracles of Engineering

FTC Team 365

2018-19 Control Award Submission

Introduction

Throughout the design of our robot, we have kept one universal theme in mind.

User Friendliness: The measure of how robust, simple, easy to maintain, and easy to use a robot is.

To accomplish this goal of user friendliness in Autonomous and TeleOp, we have tried to keep the number of important components on the robot (sensors, motors, etc...) to a minimum while still vying to accomplishing our goals in mind. The result has been a robot that places more importance on intricate algorithms than sensors.

Our robot does utilize a good number of sensors, but wherever one can be omitted (for example: a camera rather than a color sensor), we take that option. This results in less environmental variables that can impact robot performance, as the robot relies on its algorithms and math to do computation in the place of sensors that could sometimes provide faulty data.

When driving the robot, we try to keep controls as simple as possible to allow the driver to focus on making important decisions rather than be distracted or bothered with the controlling of the robot.

Along with programming for the sake of the robot in competition, we have also programmed for the sake of learning (such as creating our own Neural Network!) to involve ourselves in other forms and kinds of programming. For an explanation on our thought process & more experimental procedures, view the Additional Summary Information. Also, below most titles will be a listing of notebook pages grouped together by what stage in the development process they show.

The 6 main sections are as follows:

1. Autonomous Objectives
2. Sensors Used
3. Key Algorithms
4. Driver Controlled Enhancements
5. Autonomous Program Diagrams
6. Additional Summary Information

Table of Contents

1.0 Autonomous Objectives

2.0 Sensors Used

3.0 Key Algorithms & Constructs

4.0 Driver Controlled Enhancements

5.0 Autonomous Routines

6.0 Additional Summary Information

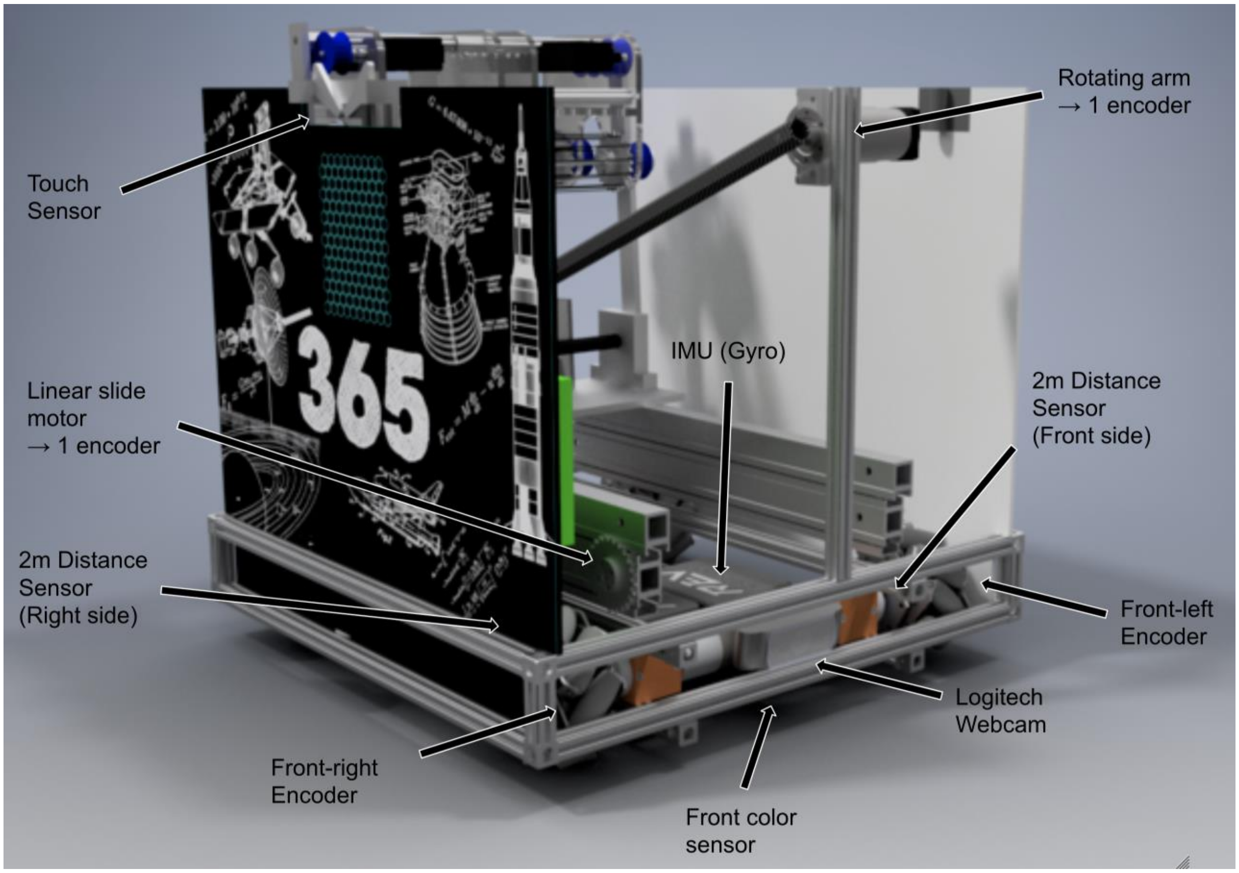
Autonomous Objectives

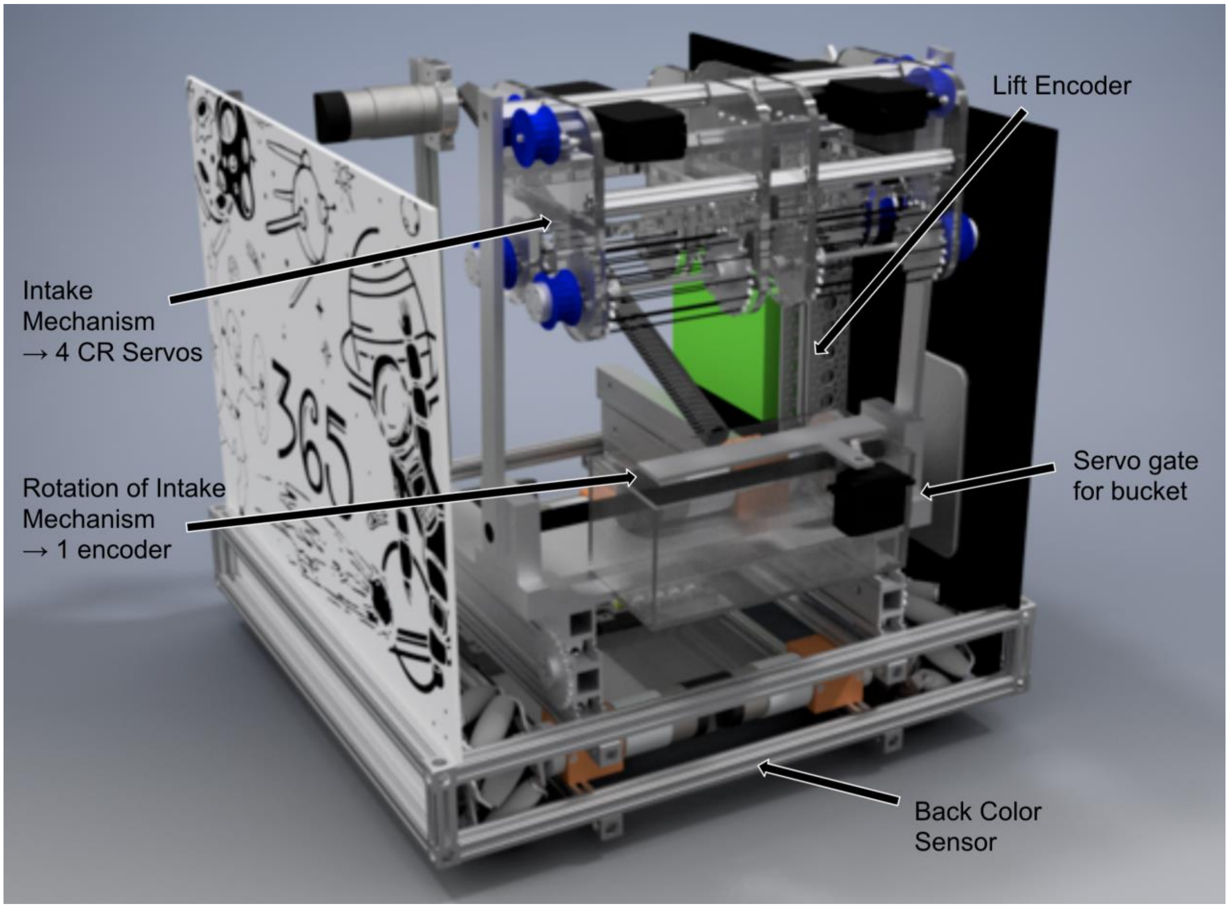
The following objectives are what we planned for in our robot's autonomous modes.

- **Autonomous Routine: (82 pts.)**
  - Landing – dropping off of the lander (30 pts.)
  - Sampling – knocking off the gold mineral (25 pts.)
  - Sampled gold mineral placed in depot (2 pts.)
  - Claiming – dropping the Team Marker in the depot (15 pts.)
  - Parking – ending autonomous in the crater (10 pts.)
- **Algorithmic & Programming Objectives:**
  - Establishing Field Grid & MOEPS (MOE Positioning System)
  - Localization
  - Multithreading
  - Accurate Turning Methods
  - Accurate Pathfinding with Intelligent Algorithms
    - Pathfinding Error Correction
    - "Rotational Symmetry"
    - Realignment
  - Error Correction & Fallback Plan B Routines

Although not completely relevant in explaining the controls and actions of the robot, we included additional algorithms and details in our programming process that we felt to be of importance in the *Additional Summary Information*.

Sensors Used




Lift Encoder
Intake Mechanism → 4 CR Servos
Rotation of Intake Mechanism → 1 encoder
Servo gate for bucket
Back Color Sensor

Encoders (3)

2: Encoders placed on motors involved with the robot’s mecanum drive. One encoder is on the front-left wheel, while the other is on the front-right wheel.

1: Encoder placed on the lift motor involved with dropping/hanging. The encoder is used for precise, controlled motion of the lift motor.

1: Encoder placed on the motor controlling the linear slide of our harvester. The encoder is used for making sure the slide does not swing out uncontrollably during autonomous.

1: Encoder placed on the motor controlling the bucket where minerals are deposited in the lander. The encoder is used for precise, controlled motion of the motor.

Inertial Measurement Unit (IMU) (1)

1: IMU build into the REV Expansion Hub. This IMU is effectively a gyro sensor, with the capability to measure rotation on 3 axes. We primarily use the horizontal axis, or the one that measures rotation parallel to the ground for accuracy in any turns or rotational movement of the robot.

Logitech Webcam (1)

1: Logitech Webcam used in the front of the robot. Using this rather than a phone camera allows for the phone to be safely protected within the robot, making sure that nothing goes wrong during the match. The Logitech Webcam is used for recognizing the Vumarks.

REV 2m Distance Sensor (2)

1: REV 2m Distance Sensor placed on the front side of the robot, facing the forward direction. The sensor is primarily used for telling distance away from the walls of the field.

1: REV 2m Distance Sensor placed on the right side of the robot, facing the right direction. The sensor is primarily used for telling distance away from the walls of the field.

Color Sensor (2)

1: Color sensor placed on the front of the robot, facing the ground to align with lines during autonomous.

1: Color sensor placed on the back of the robot, facing the ground to align with lines during autonomous.

Touch Sensor (1)

1: Touch Sensor placed near the top of the Tetrix channel used for holding the linear actuator used in dropping/hanging. The sensor is used for safe and automated resetting of the lift in hanging. Since the lift has to have the same starting point across all robot autonomous modes, a standard starting point is important.

Key Algorithms & Constructs

Field Grid & MOEPS (MOE Positioning System)

Conceptualization and implementation: C7, C8, C13, C14

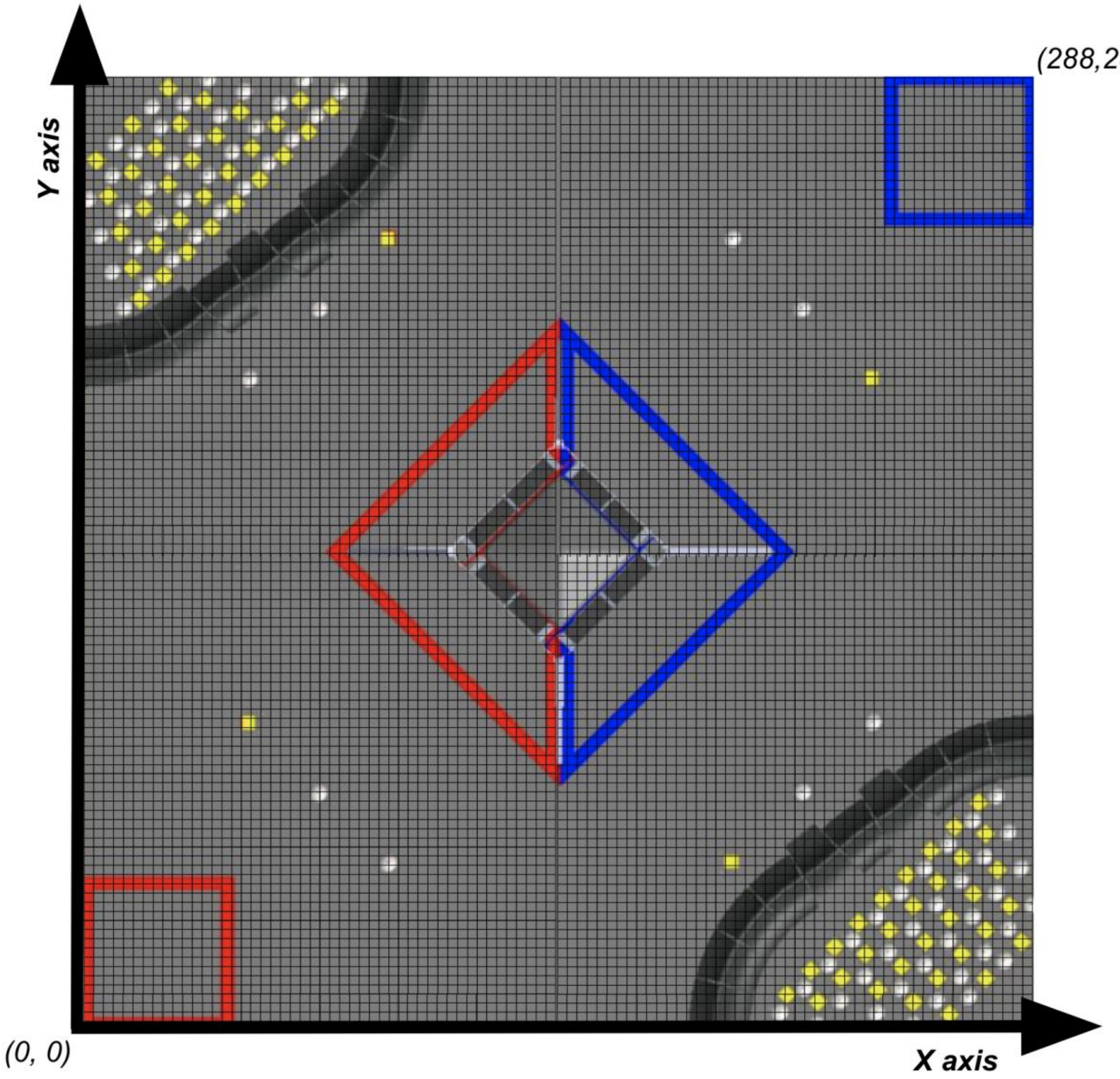
Due to the importance of the two positioning systems described below in our programming structures, we have affectionately coined the term MOE Position System, or MOEPS, to describe the systems.

Many of the following approaches and techniques we use rely upon a grid of (x, y) points. To form this grid, we divided up the field into a grid on the Cartesian plane, from (0, 0) to (72, 72).

A real field is 12ft. x 12ft., and we divided up the field into a 72 x 72 grid, where each MOE unit = 2 inches.

12 feet = 144 inches = 72 MOE units

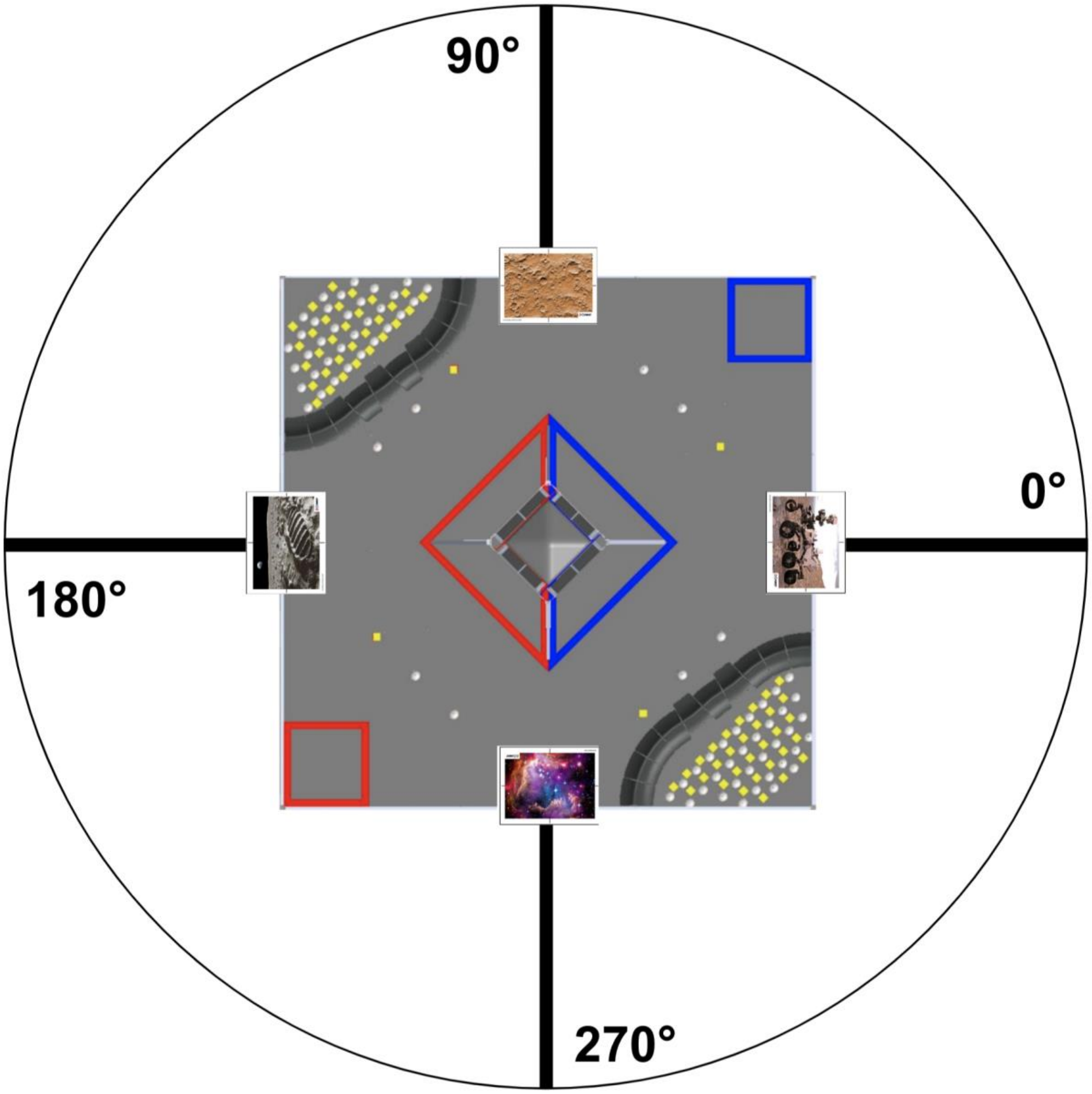
Additionally, the corner of the red depot was used as (0, 0) of the grid. Any of the four corners could have been used as (0, 0), so it was an arbitrary decision to choose the red depot.



To ease our programming, we made a Java class called PointMap to hold all important (x, y) points on the field with English names. While programming, we were able to refer to these names rather than the actual (x, y) coordinate. The following places were labelled as important:

Lander, Red Side Crater, Blue Side Crater, VuMarks, Field Corners, Sampling At Red Crater, Sampling At Blue Crater, Sampling At Red Depot, Sampling At Blue Depot, Red Depot, Blue Depot

Along with a positional (x, y) global map, we wanted to create an orientational global map to establish a consistent angle at any point on the map. The angle map was modeled off of the *Unit Circle*, which was used as a standard for marking angles on the map.


90°
0°
180°
270°

Localization

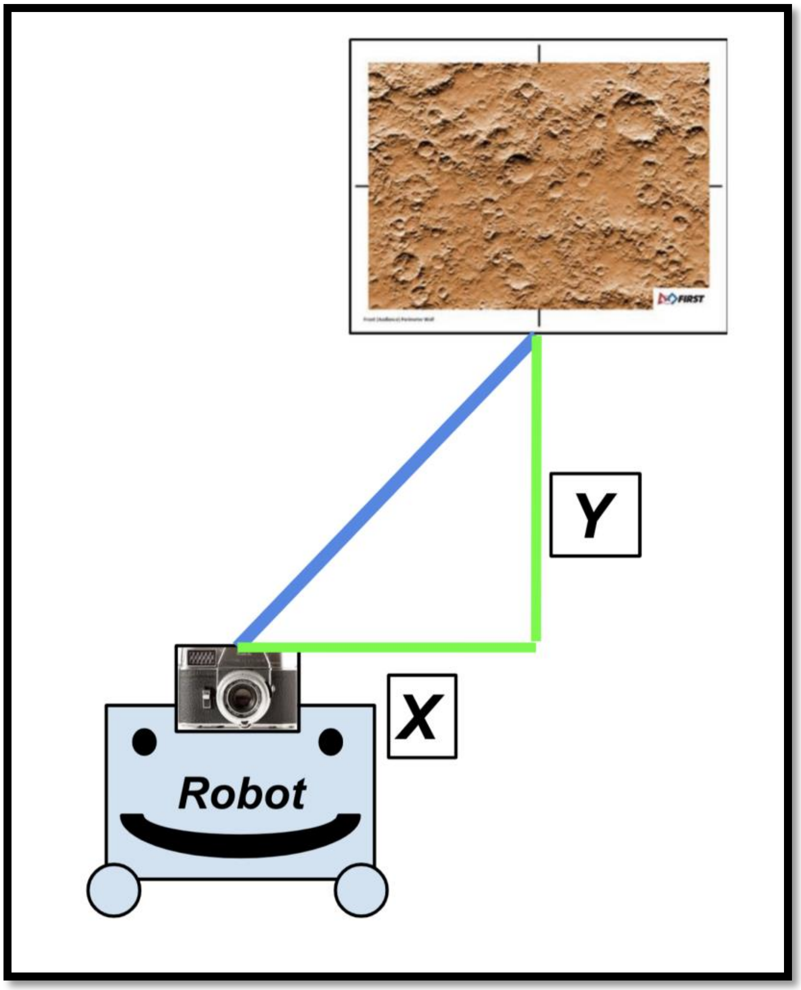
See engineering notebook entries: C42, C43, C146, C147, C148

In terms of our programming team, localization means to find out the robot's exact global (x, y) position on the field. In this case, the robot would have to find out its global (x, y) position on the **MOEPS global field grid** (check *Defining The Field Grid*). To accomplish this, we make use of the **Vuforia** image recognition technology and the **REV 2m Distance Sensor**.

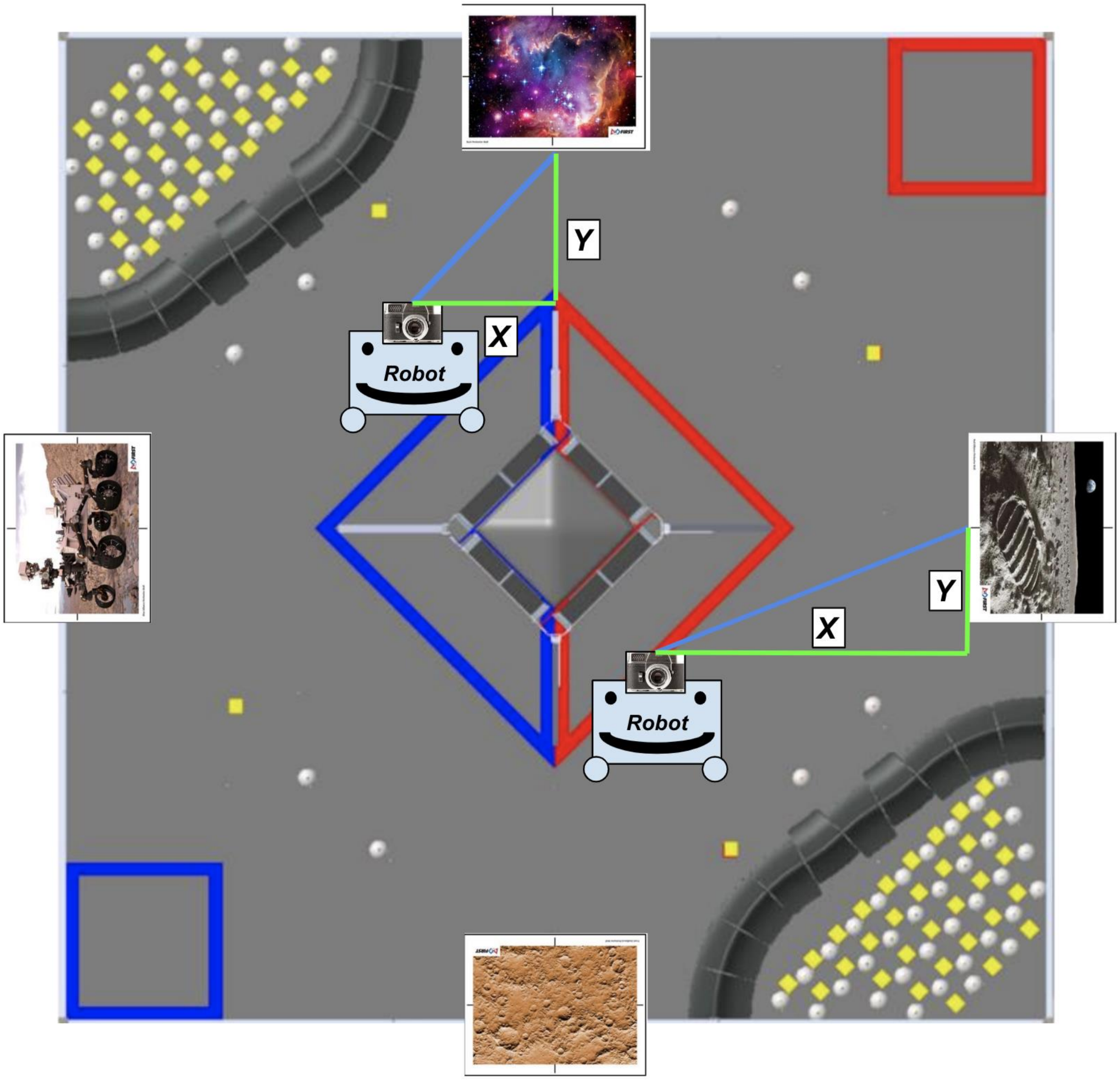
VuMark Localization

When the robot's webcam sees a VuMark, the following steps are taken:

1. Extrapolate horizontal (x) and vertical (y) distance from the VuMark using Vuforia
2. Scale the x, y distance into our 2-inch units – this is done by multiplying the values by the scalar 1/50
3. Depending on the VuMark, subtract the x, y values as appropriate – each VuMark has a distinct x, y position on the field, so subtract the robot's local x, y position from the VuMark from the VuMark's global position



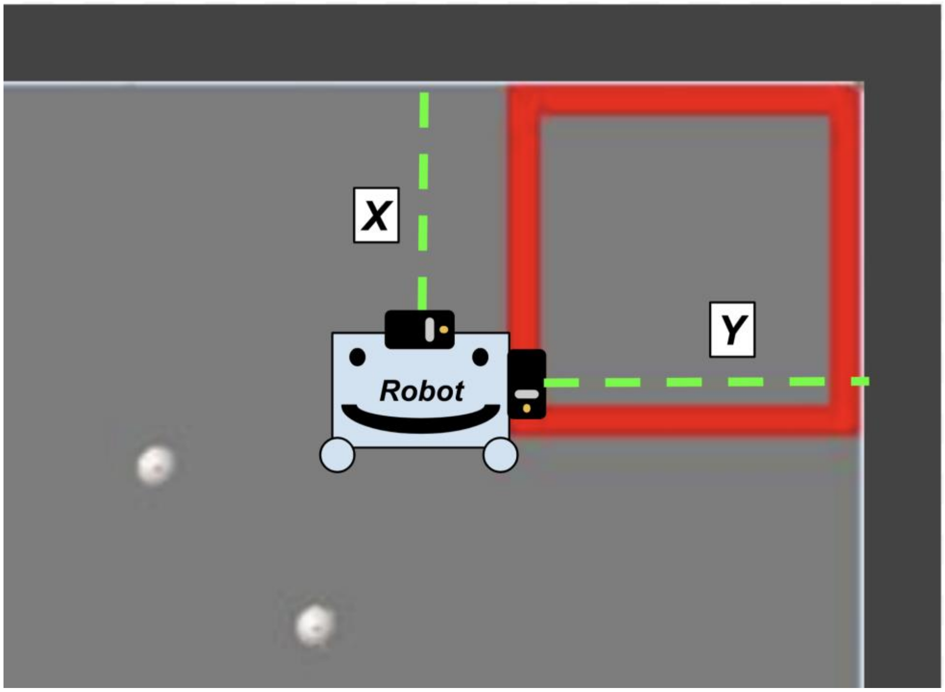
VuMark Localization on the field



Distance Sensor Localization

Localization using the distance sensor is similar to the VuMark Localization method, but is less reliable due to inaccuracies and errors that occasionally occur in the distance sensors. It used when no VuMarks are available and localization is necessary.

In this case, the following steps are taken: 1. Get vertical and horizontal distance in inches from wall with distance sensors 2.



Subtract inches from wall (x, y) position

3. Resulting (x, y) coordinate is the robot’s global location

Multithreading

Implementation of lift mechanism: C140

Implementation of realignment: C58, C59

Multithreading is a technique by which a single set of code can be used by several processors at different stages of execution. In other words, a program can have multiple sets of instructions running at the same time. With multithreading, the robot is able to do *more than one task* at any given time. Since our robot is trying to accomplish all standard autonomous points, time is often cut close to 30 seconds.

Without the use of multithreading, the robot's autonomous routine would have to speed up its motors significantly to meet the allotted 30 seconds. This speeding up results in less accuracy, resulting in an autonomous that is more prone to error.

Although processes like Vuforia and TensorFlow may run on separate threads, we intentionally use our own threads or pull from other threads for the following purposes:

- Bringing down the lift mechanism used for dropping/hanging
- Pathing algorithm realignment (see *Vuforia Listener* in *Additional Summary Information* for more details)

We also used **Atomic variables** for thread-safe operation. When a global variable is dealt with between 2 or more threads, there is always the danger of it leaking data when operations on it are done at the same time. Since using a raw variable without synchronization or any other standard is considered bad practice, we decided to use Atomic variables for thread-safe operation. This way, when communicating between the Main Robot thread and the Vuforia thread, we can guarantee that no strange behavior in autonomous occurs because of loss of data between the two threads.

Turning Methods

Conceptualization and implementation: C31, C32

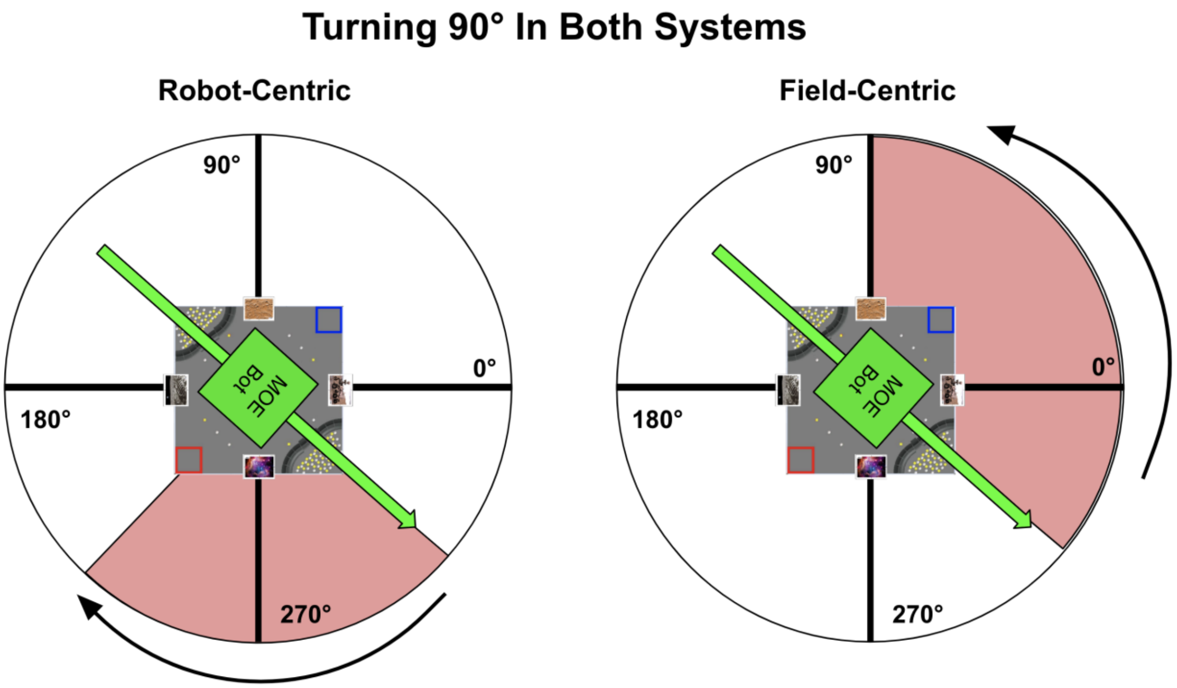
Turning in autonomous has to be precise to the degree for repeatable results, which is why turning is dictated by the **IMU** sensor built into the ***REV Expansion Hubs****.* Instead of turning by time, we turn by setting the powers the motors and simply wait for the **IMU** to indicate that we are within the correct angle.

Field-Centric Turning & Robot-Centric Turning

In **field-centric turning,** the **MOEPS angle map** described above (see *Field Grid & MOEPS*), the robot turns to a given global angle on the field.

In **robot-centric turning** the robot turns to a given angle relative to itself.

Robot-centric vs. Field-centric Turning:



As shown in the diagram above, the robot turns 90° directly to the right in the robot-centric turn, while the robot is turning to the 90° mark in the field-centric turn. No matter the orientation, the robot will always turn to the same 90° mark in field-centric turning.

Jump Point Search / A* / Dijkstra's Pathfinding Algorithm

*Conceptualization and implementation of old linear pathfinding algorithm (**not used on robot**):* C42, C43

Conceptualization and implementation: C46, C47

Implementation and testing: C50, C51

Debugging and gradual improvements: C55, C56, C58, C59, C64, C76, C93

Radius and size reductions: C97, C98, C99

2nd Stage Debugging: C104, C105

8-Directional Movement: C128, C129

3rd Stage Testing: C146

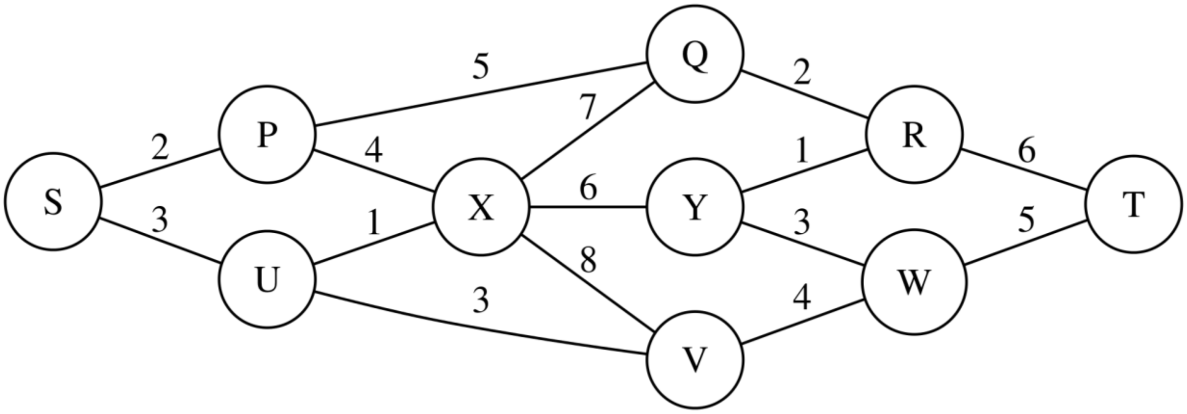
"Rotational Symmetry": C150, C151

Introduction

The purpose of the algorithm is to allow the robot to dynamically figure out how to reach its destination. In many autonomous pathings, whenever there is a slight disturbance, the autonomous fails to finish. Rather than explicitly giving the robot a path to follow, the robot is given an end destination. Through localization, the robot figures out its (x, y) coordinate on the field and calculates on its own how to reach the destination.

The *A* (pronounced A Star)* and *Jump Point Search Algorithms* are similar to the popular *Dijkstra's Algorithm*, which is used for finding the shortest paths between nodes in a graph. The primary difference between Dijkstra's and the other two is that the pair utilize a "heuristic function", or an approximation function, to approximate a faster solution to Dijkstra's algorithm. Dijkstra's algorithm checks many more cases than the A* Algorithm, therefore taking longer to arrive at a similar answer. Since the field we are using is 288x288 (82944) nodes, we wanted to guarantee that processing speed would be fast. The algorithms commonly deal with graphs shown like the one below, but had to be specially adapted in our case to work with a 2D grid.

Visual representation of traditional graph in computer science:



To account for processing speed & time, Dijkstra's Algorithm has a worse case time complexity (when using lists) of **O(N^2)** where N = number of nodes on the graph, while A* *generally* has a time complexity of **O(b^d),** where b = branching factor and d = depth of the solution on the search tree. However, both of these algorithms have a very slow runtime in certain circumstances, taking over 20 seconds to run. This is unacceptable when run in autonomous, which only has a period of 30 seconds. The Jump Point Search algorithm is an optimized version of the A* pathfinding algorithm that consistently brought our runtime below 2 seconds.

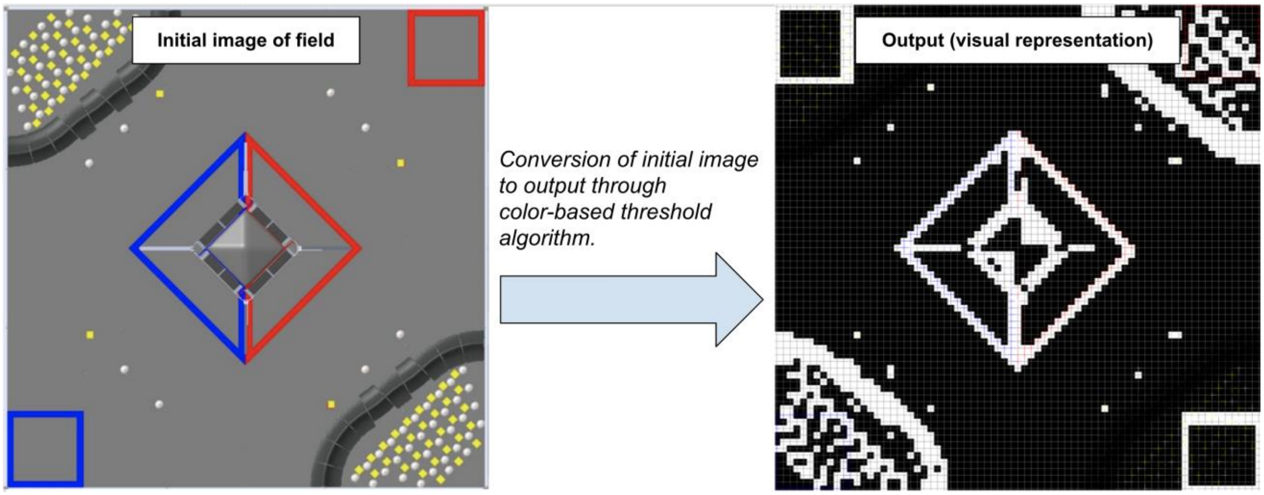
Note that the time complexity of A*/Jump Point Search is worse when using a very expensive heuristic cost function, but we are using the simple Euclidean distance, or the distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

With the pathfinding algorithms, the robot is able to move in **8 directions** to go from point A to point B. The directions are labelled as follows: **North, Northeast, East, Southeast, South, Southwest, West, and Northwest.**

Setup

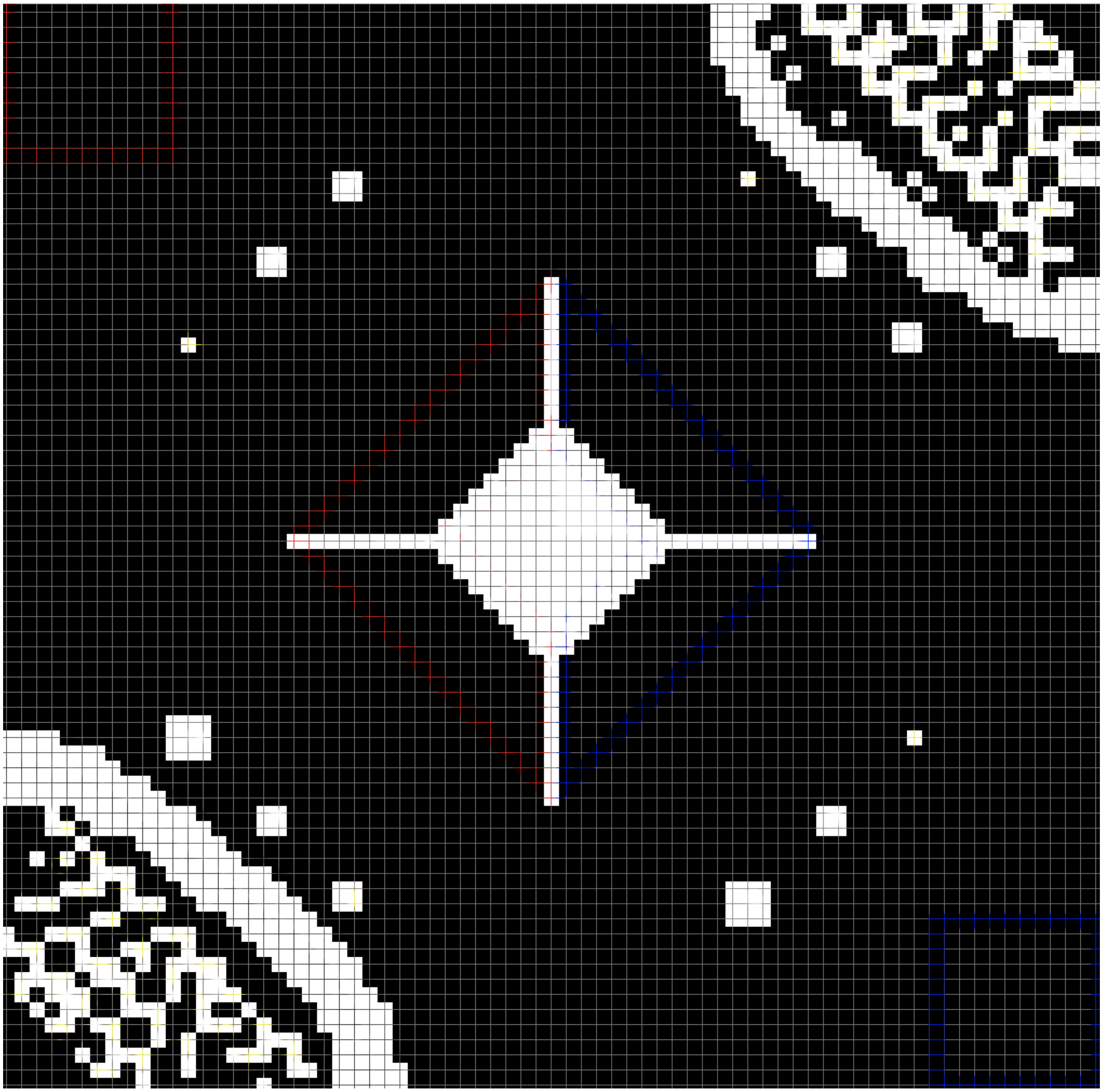
To utilize the **pathfinding algorithms**, we needed to first setup a graph. To accomplish this, we took a 2D image of the field from Game Manual 2. After that, we wrote a **Python** script (utilizing the *PIL imaging library*) to go through the image, converting it to points we deemed as barriers (white) and points we deemed as free space the robot could travel on (black). This conversion was done through a color-based threshold. In essence, the gray parts of the map were free space while the other colors were barriers. The output was an image with the converted points as well as a 288x288-*dimensional array* that we would be able to use as our graph for the pathfinding algorithms. Also, the image was flipped because we wanted [0,0] of the 2D array to be the corner of the red depot, and [287,287] of the 2D array to be the corner of the blue depot.


Initial image of field
Conversion of initial image to output through color-based threshold algorithm.
Output (visual representation)

Mapped FTC Field (Visual Representation of Array)

1 (or white) = point a robot cannot travel on

0 (or black) = point a robot can travel on



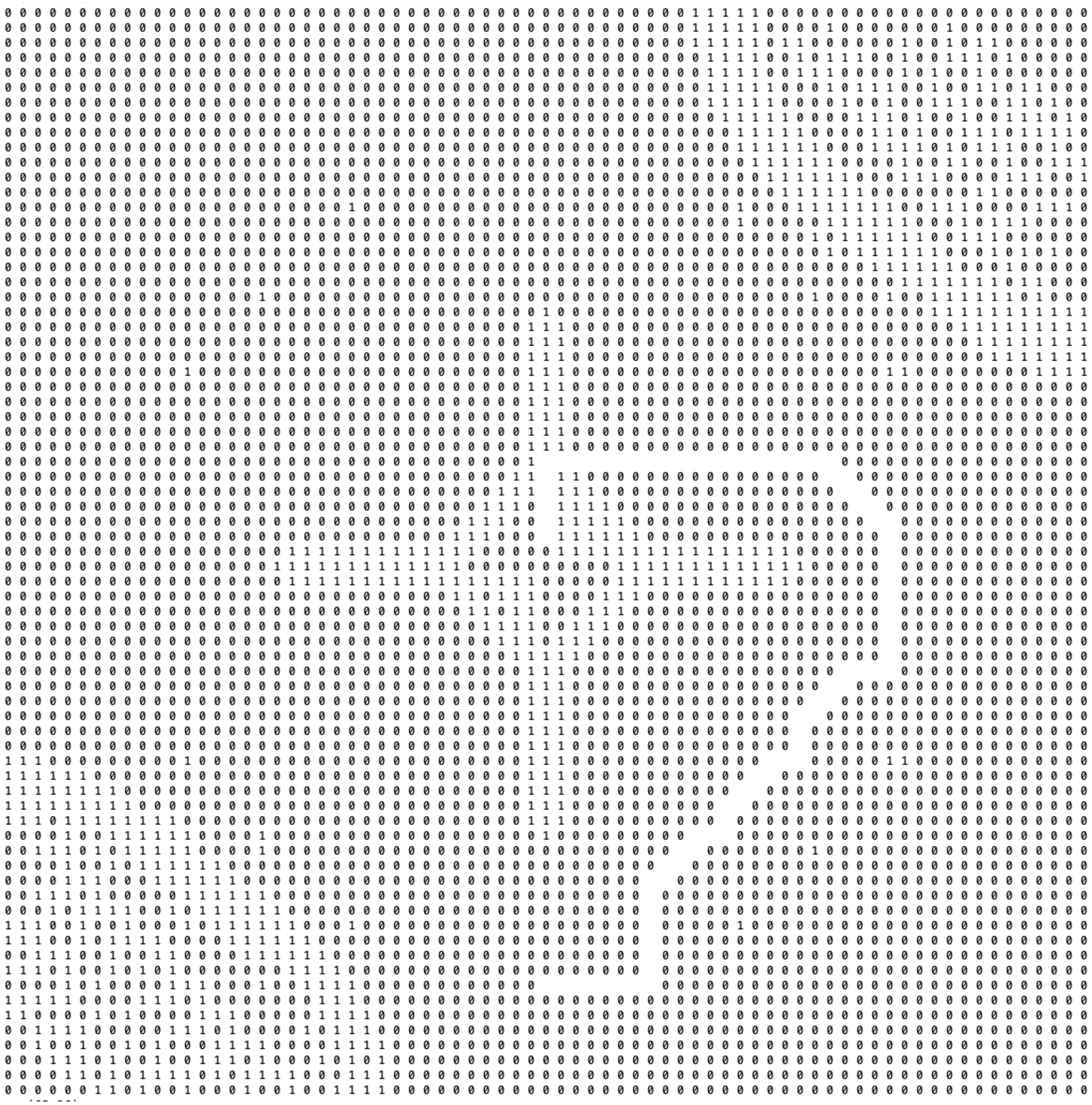
The original conversion had some errors, because places (depot, lines near the lander, etc...) were marked in white when they should have been open space. To fix this, we manually changed some values in the array. Since most of the conversion work was done by the Python script, this only took a few minutes.

The above image is what the output array looked like visually. The barriers shown in the image above are represented by 1s, while the free space shown in the image above are represented by 0s.

Implementation

We implemented the algorithm in Java, making a separate class to handle the calculations. To verify that we wrote the algorithm correctly and be able to make predictions on robot movements, we made a simulation to show the pathfinding algorithm’s path from any Point A to Point B visually:

Initial Simulation

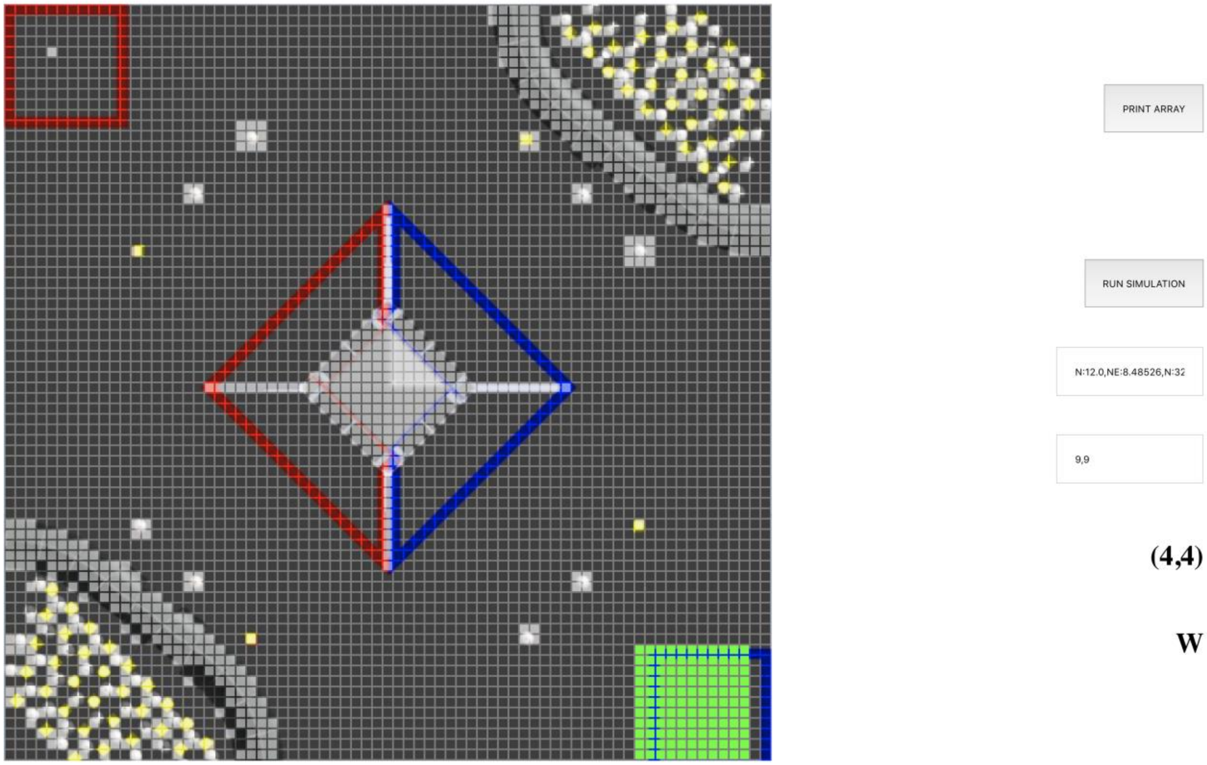


```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1                                         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1   1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1   1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0   1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0   1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0   1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0         0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0       0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0                 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

As complications with the algorithm increased, there was a need for a better simulation that more accurately depicted the algorithm in action. Below is a screenshot of an active simulation that shows the the robot (the green square), step by step, moving through the field.

Screenshot of Final Simulation

Video of live simulation: https://1drv.ms/v/s!AqOPfHs4_986ihlsFJLdiAYHtxG2



N:12.0,NE:8.48526,N:32

9,9

(4,4)

W

To now use the algorithm in practice, we had to convert the results into a usable format by writing an algorithm to do so.

Path Conversion Algorithm

(Input) - Original Pathfinding Results: A series of points describing each point to go from point A to point B. *For example, getting from (0,0) to (5,5) could be:* (0,0) --> (0,1) --> (1,1) --> (2, 1) --> (3, 1) --> (3, 0) --> (4, 0) --> (4, 1) --> (5, 1) --> (5, 2) --> (5, 3) --> (5, 4) --> (5, 5)	**(Output) - Usable Results:** The number of inches in each direction the robot has to go, in order (each unit is 2 inches). *For example, getting from (0,0) to (5,5) could be:* FORWARD 2 in. --> RIGHT 6 in. --> BACKWARD 2 in. --> RIGHT 2 in. --> FORWARD 2 in. --> RIGHT 2 in. --> FORWARD 8 in.

The robot first turns towards the **90° mark** described in the **MOEPS global angle map (**the direction facing the Crater/Mars VuMark). The results from the pathfinding algorithm would then be translated into movements for the robot based on encoder ticks. The result of this extensive process is a robust and repeatable movement system that allows the robot to figure out its own path when given two points on the field. This simplifies the process of adjusting and programming autonomous, as well as allowing for a more robust and dynamic movement system.

Pathfinding Algorithm Error Correction

The pathfinding algorithm worked perfectly well in theory, but in practice, there were a few issues we had to fix in order of importance.

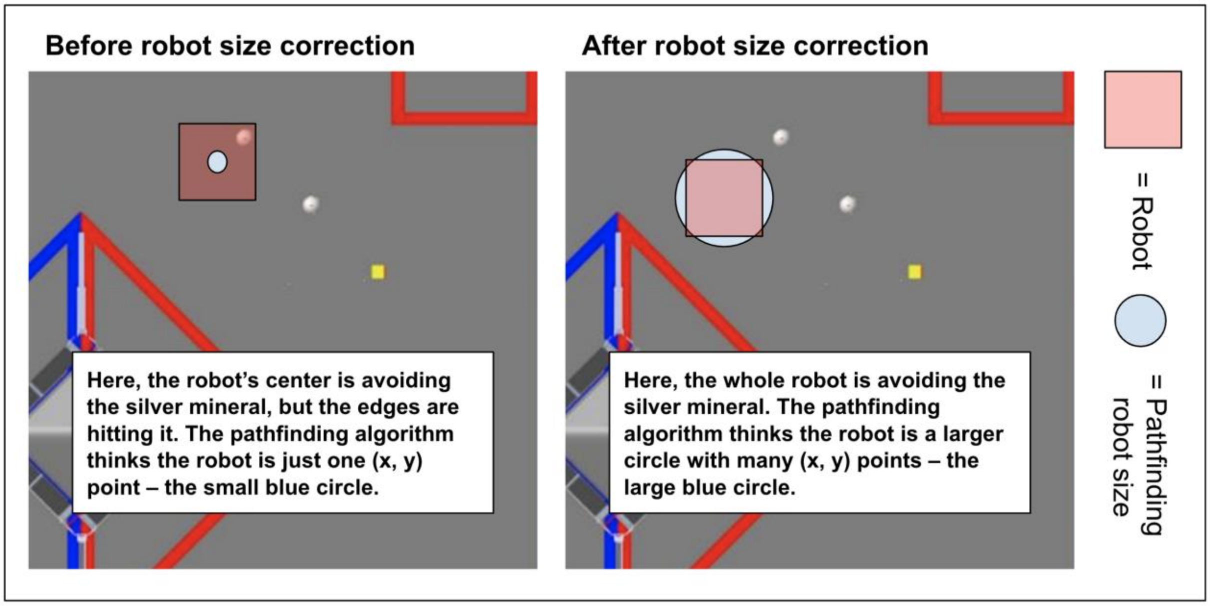
1. The **A* algorithm** treated the robot as a single (x, y) point on the **global 72x72 grid** while the robot actually comprised at least a circle of many points with radius about 10 inches. This led to the edges of the robot crashing into parts of the field (lander, crater, sampling, etc...) while its center thought it was following the pathfinding algorithms as a single small point.
2. While moving, the robot would turn slightly off angle. It would be not exactly at its end destination due to slight turning while making up, down, left, and right movements.

Error #1 – Size Corrections

Conceptualization and implementation: C55, C97

Second iteration: C98, C99

To fix this error, we modified the size of the robot in the algorithms. Instead of treating the robot as a single point, we treated it as a collection of multiple points – when put together, these points would form the robot rather than one small point.



Here, the robot's center is avoiding the silver mineral, but the edges are hitting it. The pathfinding algorithm thinks the robot is just one (x, y) point – the small blue circle.

Here, the whole robot is avoiding the silver mineral. The pathfinding algorithm thinks the robot is a larger circle with many (x, y) points – the large blue circle.

Error #2 – Turn Corrections

To fix this error, we took the IMU sensor's horizontal angle before the robot followed the A* algorithm. We then constantly tracked the gyro sensor's angle while the robot followed the A* algorithm.

If the IMU's angle strayed by more than 2°, the robot self-corrected itself back to the correct angle by again utilizing the gyro to turn back into position.

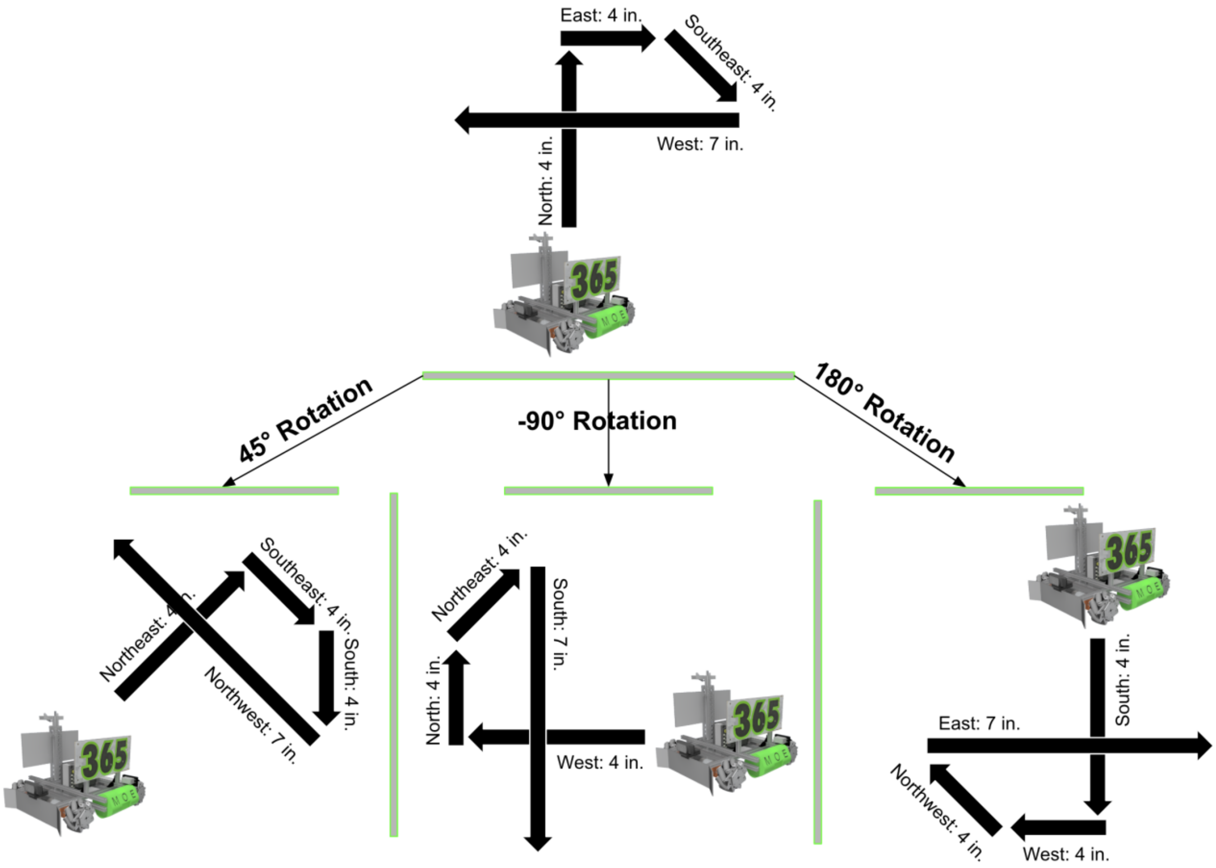
"Rotational Symmetry"

Conceptualization and Implementation: C150, C151

Another feature that we added to the pathfinding algorithm is the idea of "rotational symmetry". In other words, a given set of output instructions can be rotated by certain number of degrees while still preserving the relative directions of each movement.

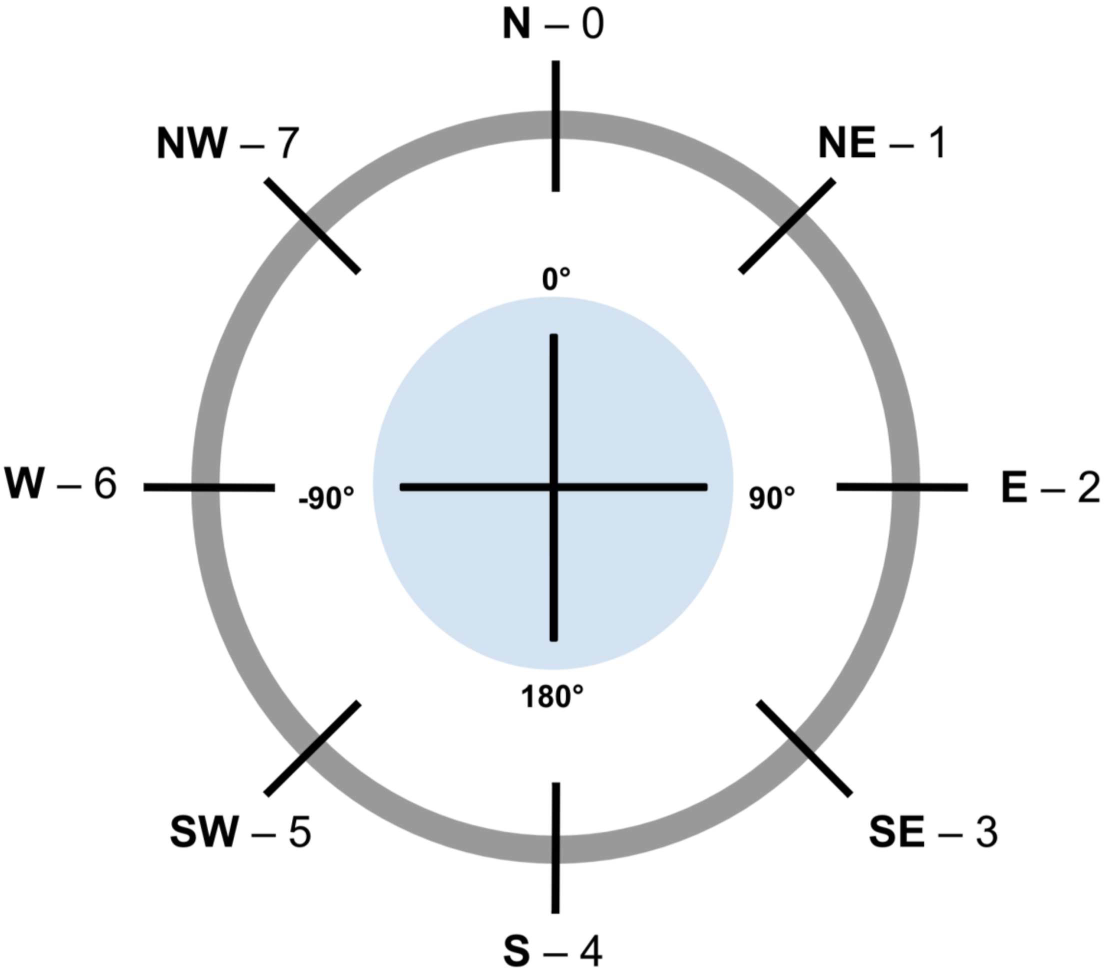
The purpose of rotational symmetry is to allow for optimizations in accuracy and speed of robot movements. Since moving forwards and backwards is always faster than strafing, rotational symmetry allows the robot to take a pathing from the Pathfinding Algorithms and rotate the pathing instructions. The robot can then quickly turn and apply the rotated instructions to allow for more forwards and backwards movements, resulting in a more robust movement.

Examples of rotations done on set of output instructions:



The algorithm is accomplished by setting a numerical value to each of the directions in clockwise order. North all the way around to Northwest is numbered from 0 up to 7. This simple numbering pattern makes rotation much simpler than writing each direction's rotation to its right. To rotate a direction, divide the angle needed to rotate by 45° and add it to the number. In the case a value goes above 7, it is wrapped back around to start at 0 (ie: 9 becomes 2).

Visualization of Direction to Number mapping, along with degrees associated with rotations:



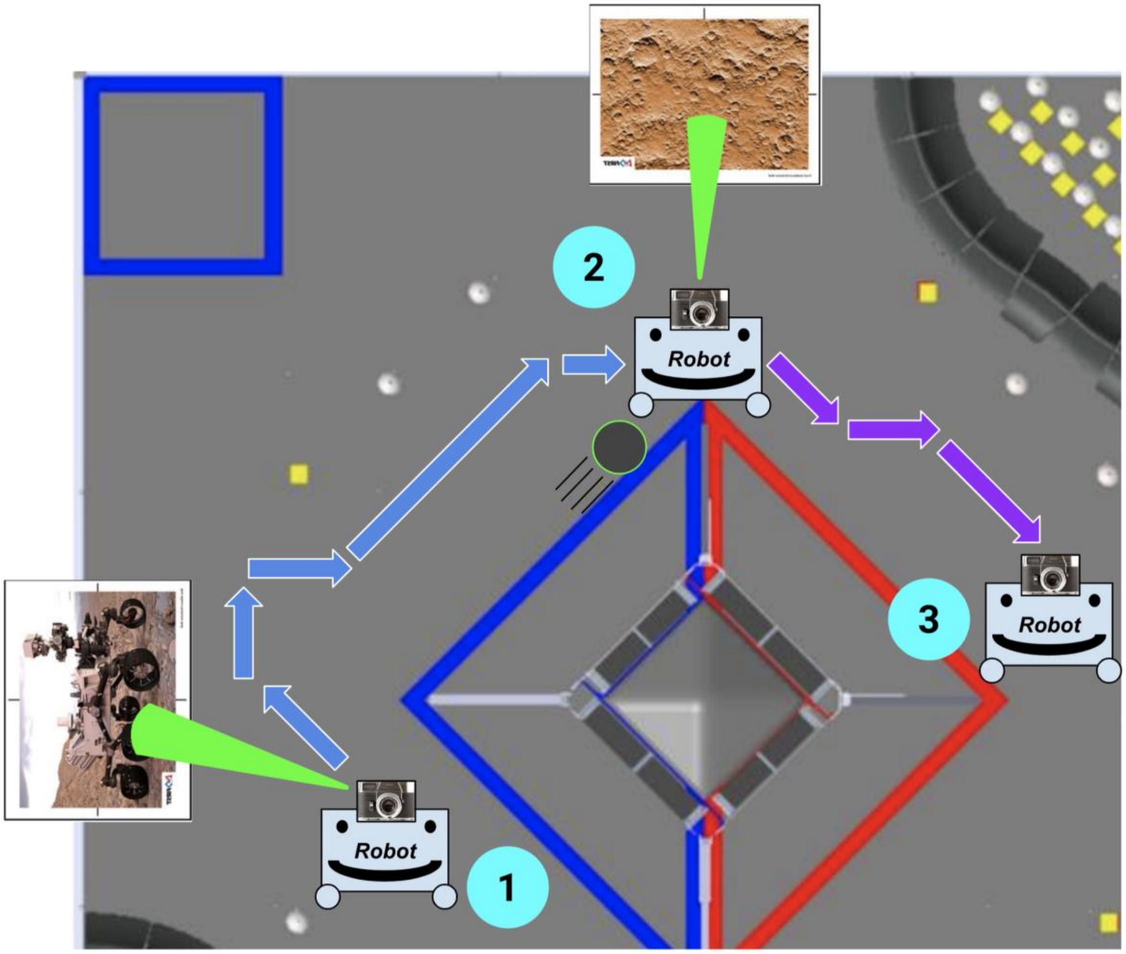
This system's simplicity becomes apparent when put into practice. For example, if an instruction says to move the robot North, the direction North's numerical value is 0. To rotate it by 90° clockwise, divide 90° by 45°, which equals 2. The 2 is added to North's numerical value, which results in 0+2 = 2. The 2 corresponds to the East direction, which is exactly a 90° clockwise rotation from North.

Realignment

Conceptualization and Implementation: C58, C59

As there is always a chance for error, such as another robot or debris in the way of a robot, a given robot might be knocked out of its planned path. This is another application for the Pathfinding Algorithms. During the course of following a path from the algorithm, the robot is always on the lookout for a new VuMark. (see *Vuforia Listener* in *Additional Summary Information*) If its camera sees one, it stops the current path it is on and restarts its pathing at the new VuMark. If the robot ever gets knocked off of its given path and fulfils the chance that it sees a new VuMark, the robot is able to get back on path.

The diagram below illustrates this process:



1. The robot localizes off of the Rover VuMark to figure out its (x, y) point
    a. Pathfinding Algorithms calculate a path to the destination (#3)
    b. Robot follows the pathing with encoders (blue arrows)
2. The robot is knocked off of its pathing by debris

   a. A new VuMark is seen and the robot stops its original pathing (blue arrows) and relocalizes, figuring out its new (x, y) point
   b. Pathfinding Algorithms calculate a path to the destination (#3)
   c. Robot follows the pathing with encoder (purple arrows)

3. Destination is reached

Error Correction & Fallback Plan B Routines

Throughout autonomous, there is opportunity for plenty of error to occur. Since a fundamental goal of autonomous is to have consistent, reproducible results, we try to better handle some errors that may result in a deviation from any planned autonomous route.

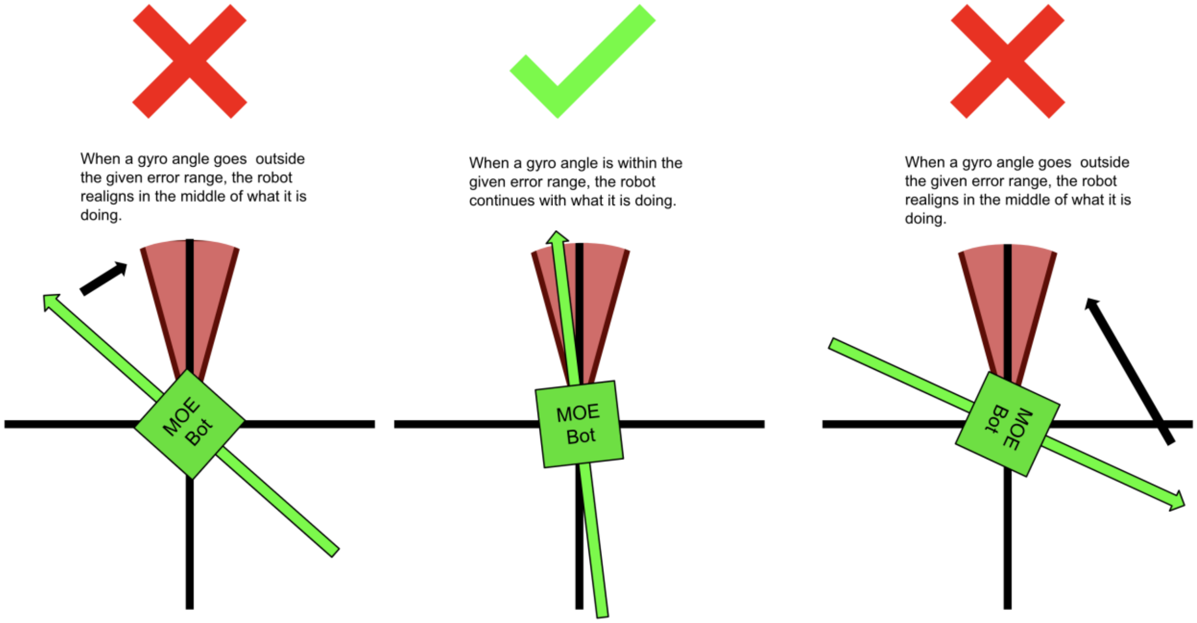
- Turn Corrections
- Distance Sensor Fallback

Turn Corrections

Implementation: C58, C59

In many of the routines and paths taken during autonomous, due to the nature of our mecanum wheels and the weight distribution of the robot, the robot gradually turns out of place. For example, when strafing normally for a period of 5 seconds, the robot could possibly turn 3° away from its starting angle. Over time, this error accumulates, and when sufficient, results in a faulty autonomous. To cut back on this, we define a given angle error range for any non-turning movement in autonomous. If the robot deviates from this error range, it interrupts what it is doing to turn back into proper position.

The diagram below shows the turn correction process.



When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.

When a gyro angle is within the given error range, the robot continues with what it is doing.

When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.

Distance Sensor Fallback

Due the occasionally unreliable nature of the distance sensors, there is a need for a fallback when they produce errors. When getting readings in the middle of autonomous, a distance

sensor sometimes gives wildly out of range readings or errors. To fix this, we add a fallback. If the sensor does not give reasonable values within 2 seconds, the robot uses a preprogrammed point to plug into the Pathfinding Algorithms rather than a more accurate point from the distance sensors.

Sampling Algorithm

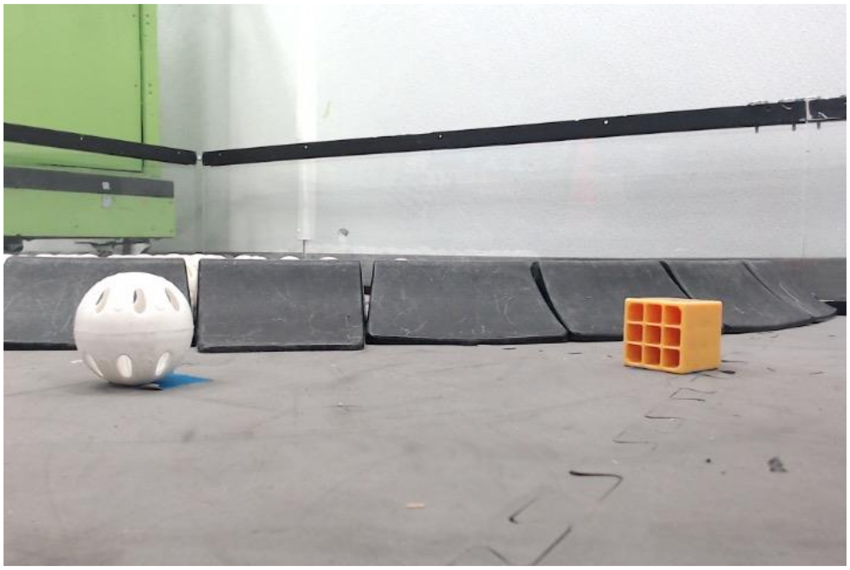
Our team tried using a variety of sampling methods, including the official TensorFlow neural network included in the SDK, creating our own neural network (see *Additional Summary Section*), and OpenCV for color detection. However, one key flaw with all of these methods was inconsistency. Even though the official TensorFlow network worked most of the time, we found that when placed on a competition field with a *yellow background* (like a gym floor), gold minerals were sometimes improperly recognized. This led to complications during competitions. Due to the flaws with the official TensorFlow network, our robot often picked the wrong mineral to sample.

To solve this issue, we wrote our own **custom algorithm** for the sake of sampling.

Steps:

1. Take camera data from Webcam

Example of camera data

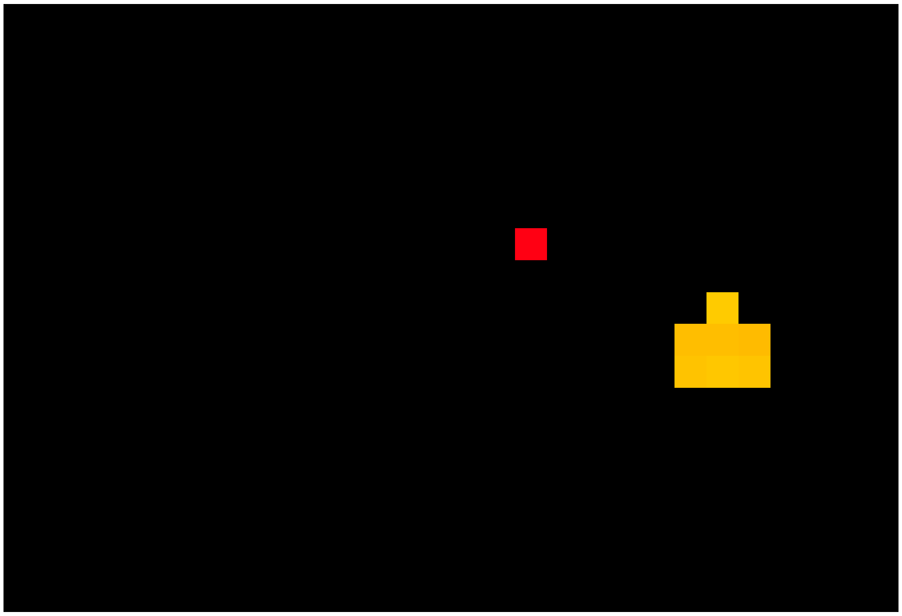


2. To remove any small splotches of yellow in the background (or any other strange color), reduce the resolution of the image.

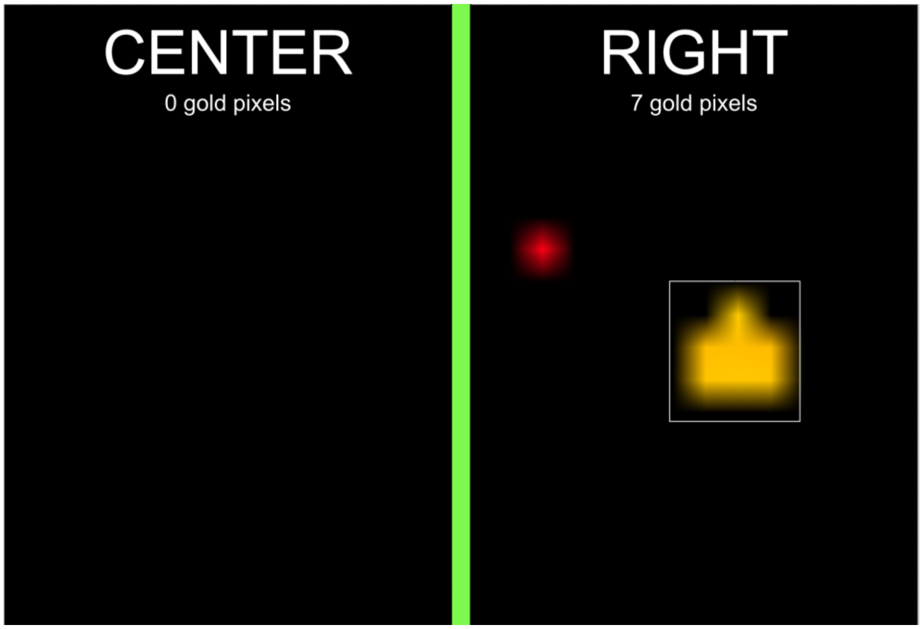
Reduced resolution of original image

3. Convert each pixel from RGB (Red, Green, Blue) color scheme to HSV (Hue, Saturation, Value) color scheme. Disregard any pixels without a high enough saturation value (< 0.5).

Visualization of image after pixels below HSV threshold are removed.



4. Compare amount of gold pixels in left and right side of image. The side that contains more gold pixels is considered the location of the gold sampling mineral. If both sides have less than 2 gold pixels, then the gold mineral is considered to be in the left location. (see *Gold Mineral Decision Algorithm* in *Additional Summary Section*)



CENTER

0 gold pixels

RIGHT

7 gold pixels

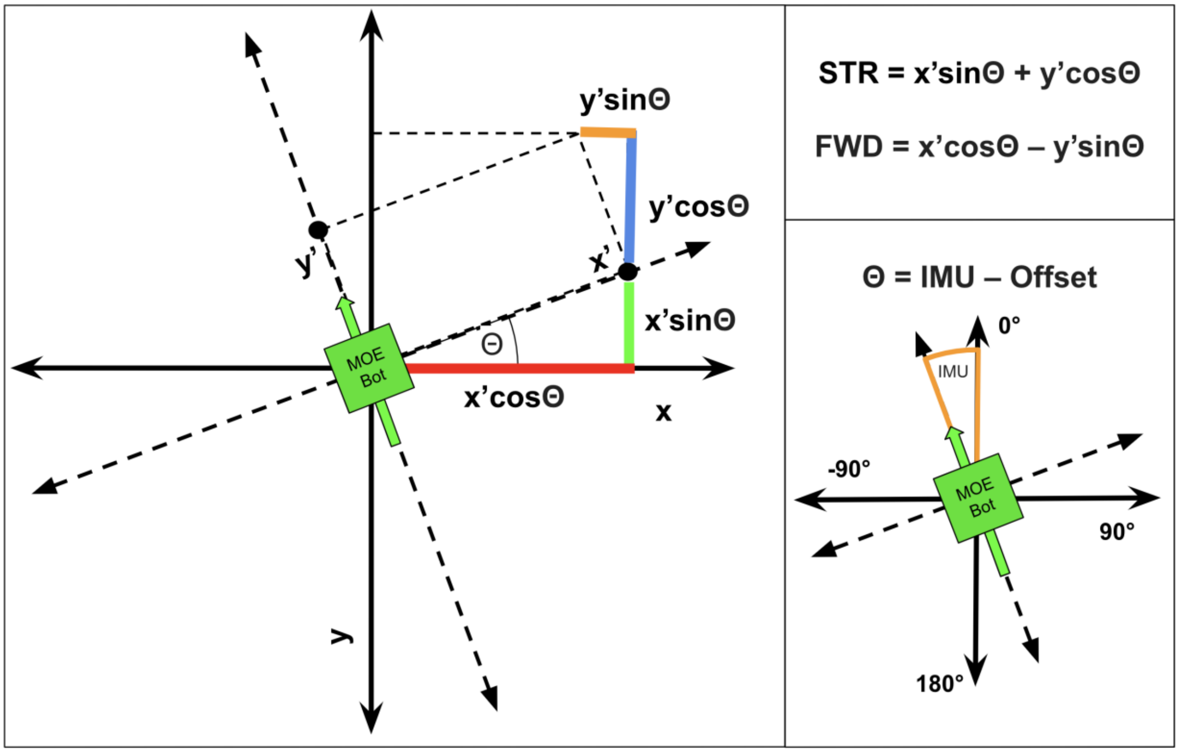
Driver Controlled Enhancements

Adjustable Field-Centric Movement

Conceptualization and implementation: C130, C131, C141

Since our robot uses a mecanum drive, it is capable of moving in any direction. Because of this, we are able to use **field-centric** motion rather than **robot-centric** motion. For the ease of the driver, we make all movements relative to the field rather than relative to the robot.

Note: In this diagram x′ = Left Joystick X AND y′ = Left Joystick Y



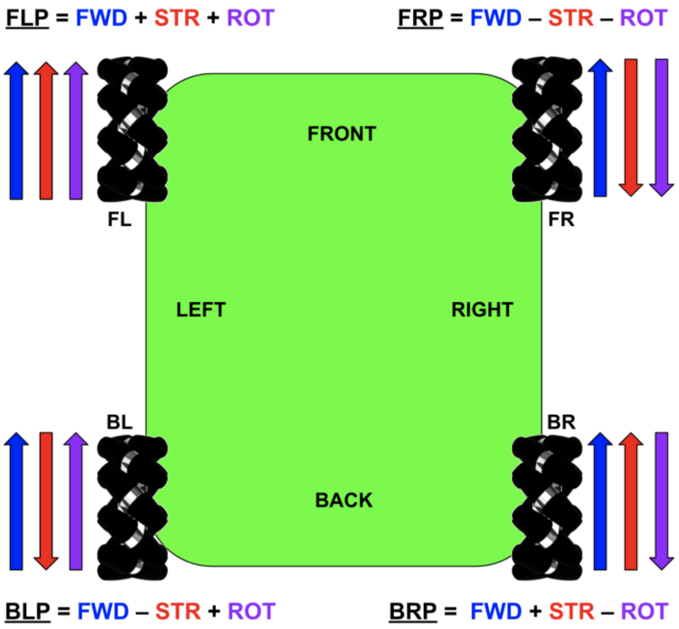
$$STR = x'\sin\Theta + y'\cos\Theta$$

$$FWD = x'\cos\Theta - y'\sin\Theta$$

$$\Theta = IMU - Offset$$

This diagram represents the **rotation of axes** that underlies the principles in field centric movement. The Θ is taken from the IMU, so that angle measurements are exact in the **field-centric** motion. Inputs for x′ and y′ are taken from the left joystick, and the right joystick x controls the rotation of the robot. Also, the offset for the IMU is to allow the driver to set a custom 0° point for the robot.

In the below diagram, the arrows indicate which direction the wheel turns when given a positive value for FWD (forward), STR (strafe), or ROT (rotation).



The corresponding values of FLP, FRP, BLP, and BRP are fed into the motors based on controller input so that field-centric movement occurs.

Lander Based Movement

Conceptualization and implementation: C141

Our robot also has “**lander specialized”** movement that allows for fine turning before hanging the robot. The D-pad allows for smaller adjustments in four-directional movement corresponding to the four buttons on the D-pad. Also, the bumpers allow for smaller adjustments in turning to fix orientation before raising the lift to hang.

Controls

Gamepad 1

Left Joystick: Field-centric movement in all directions

Right Joystick: Rotational movement (turns)

A: Reset 0° point (forward heading) for field-centric movement

X: Open Bucket Halfway

B: Open Bucket Full

Y: Toggles Tele-Op and End Game Mode

Left & Right Triggers: Moves Rotating Arm Up and Down in Tele-Op Mode and Latching Lift in End-Game Mode

Left & Right Bumpers: Fine turning (see *Lander Based Movement*)

D-Pad: (see *Lander Based Movement*)

UP → Move slowly forward

DOWN → Move slowly backward

LEFT → Strafe slowly left

RIGHT → Strafe slowly right

Gamepad 2

Left Joystick: Rotates Transition arm (wrist) up and down

Right Joystick: Moves Linear slide in an out

Left & Right Triggers: Intakes using harvester (independently-run with respective controls)

Left & Right Bumpers: Dispenses using harvester (independently-run with respective controls)

Autonomous Routines

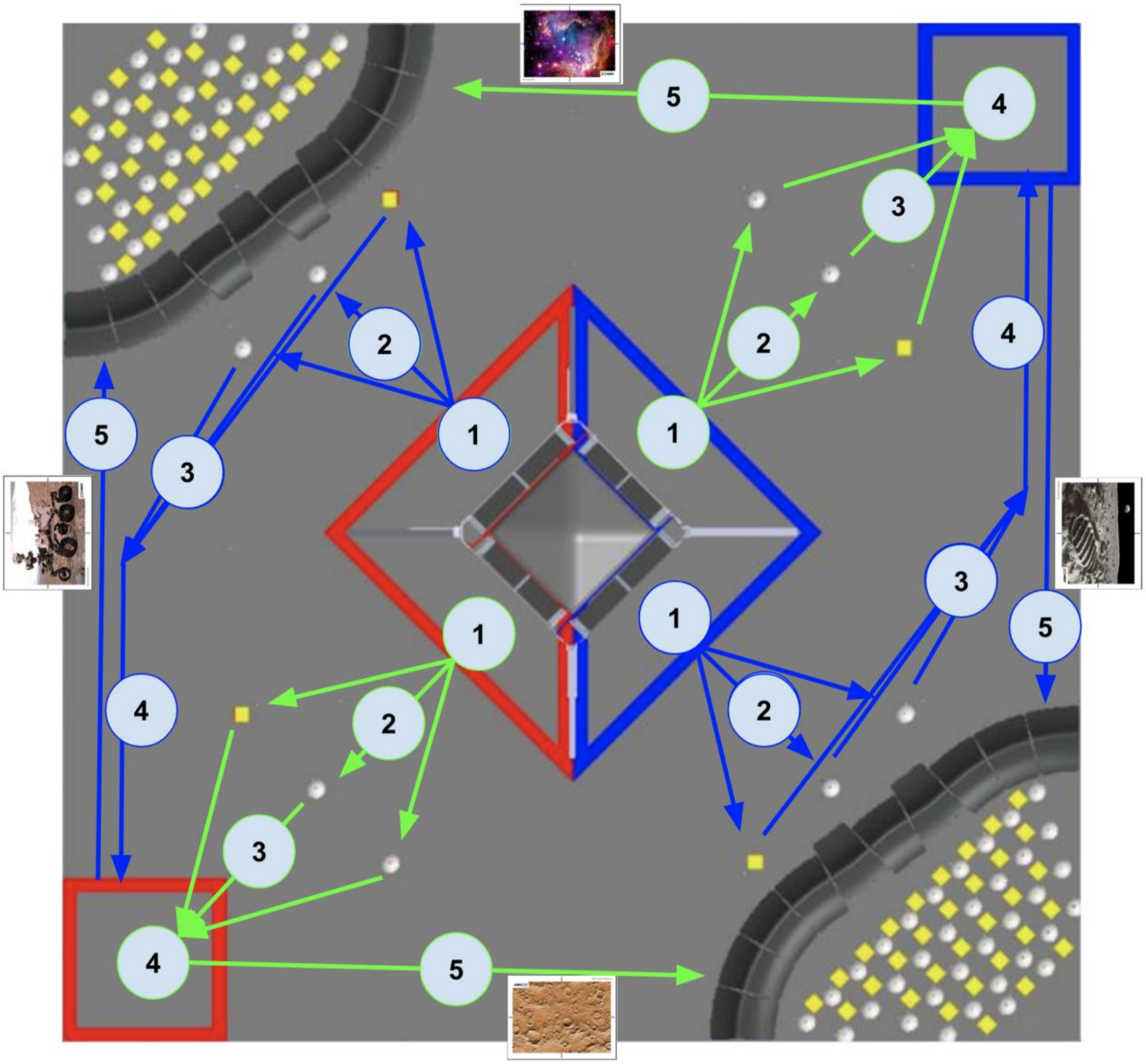
Conceptualization and testing of routine: C76

Iterative improvements, testing, and debugging: C84, C85

Testing environment fabrication: C104

2nd Stage Iterative improvements, testing, and debugging: C105, C106, C121, C122

Testing and evaluations: C140, C141, C145



The image above represents our autonomous routine for all possible starting points. Regardless of starting point, our autonomous strives to accomplish delatching, sampling, depositing team marker, and parking.. We plan on executing the following steps for each autonomous:

1. Landing & Detecting gold mineral
2. Travelling to and knocking off gold mineral
3. Travelling to depot
4. Deposit the Team Marker and leave gold mineral
5. Travel to crater & extend arm into crater

Initialization

1. Initialize motors, servos, sensors, and all other devices from the Hardware Map.
2. Initialize REV IMU sensor
3. Initialize Vuforia
4. Reset Crater Extension Arm to position 1
5. Have robot say “Finished initialization” through speakers to doubly confirm initialization

1. Landing & Detecting gold mineral (30 pts.)

1. Use linear actuator to move lift up for given # of encoder tics
    a. The robot touches the floor and the claw goes above the top of the handle on the lander
2. Using multithreading, lower the lift back to its starting position
    a. (see *Multithreading* in *Key Algorithms*)
    b. The program continues on without waiting for the process to finish because of the multithreading
3. Turn ~70° to see 2 minerals and decide which is gold
    a. (see *Sampling Algorithm* in *Key Algorithms*)
4. The robot turns the appropriate # of degrees to face the gold mineral in left, right, or center

2. Landing & Detecting gold mineral (25 pts.)

1. Move forward appropriate # of inches to knock off gold mineral from its starting position
2. Continue moving forward to safely clear other 2 silver minerals

3a. Travelling to depot (Depot Start Point)

1. Turn appropriate number of degrees to face VuMark (Rover/Moon)
2. Move forward appropriate # of inches to read data from the VuMark (distance, angle, etc...)

3b. Travelling to depot (Crater Start Point)

1. Turn appropriate number of degrees to face the depot
2. Move forward appropriate # of inches to reach the depot
3. Localize and figure out (x, y) position on the MOEPS global field grid by using Vuforia
4. Calculate and follow path to depot using Pathfinding Algorithms

4. Deposit the Team Marker (15 pts.) – If Depot Start Point, deposit gold mineral in depot (+2 pts.)

1. Turn appropriate # of degrees for front of robot to face the front/back wall
2. Localize and figure out (x, y) position on the MOEPS global field grid by using distance sensors
    a. (See *Distance Sensor Localization* in *Localization*)
    b. If the distance sensors have an error in measurement, fallback to Plan B
3. Turn appropriate number of degrees for left side of robot to face corner of field
4. Drop off Team Marker
5. Calculate path to crater using Pathfinding Algorithms
    a. ***Plan A:*** Use (x, y) position from distance sensors
    b. ***Plan B:*** Use default (x, y) position as an estimate – less accurate than Plan A
    c. (see *Jump Point Search/A*/Dijkstra's Pathfinding Algorithms*)

5a. Travel to crater & extend arm into crater (10 pts. – Depot Start Point)

1. Turn ~90° right for the back to face the crater
2. Apply 90° rotation to pathfinding algorithm results to fit new robot orientation
    a. (see "*Rotational Symmetry" in Jump Point Search/A*/Dijkstra's Pathfinding Algorithms*)
3. Follow pathing to reach crater on other alliance's side
4. Extend arm
5. Drive backwards to guarantee arm is in crater

5b. Travel to crater & extend arm into crater (10 pts. – Crater Start Point)

6. Follow pathing to reach crater
7. Extend arm
8. Drive backwards to guarantee arm is in crater

Additional Summary Information

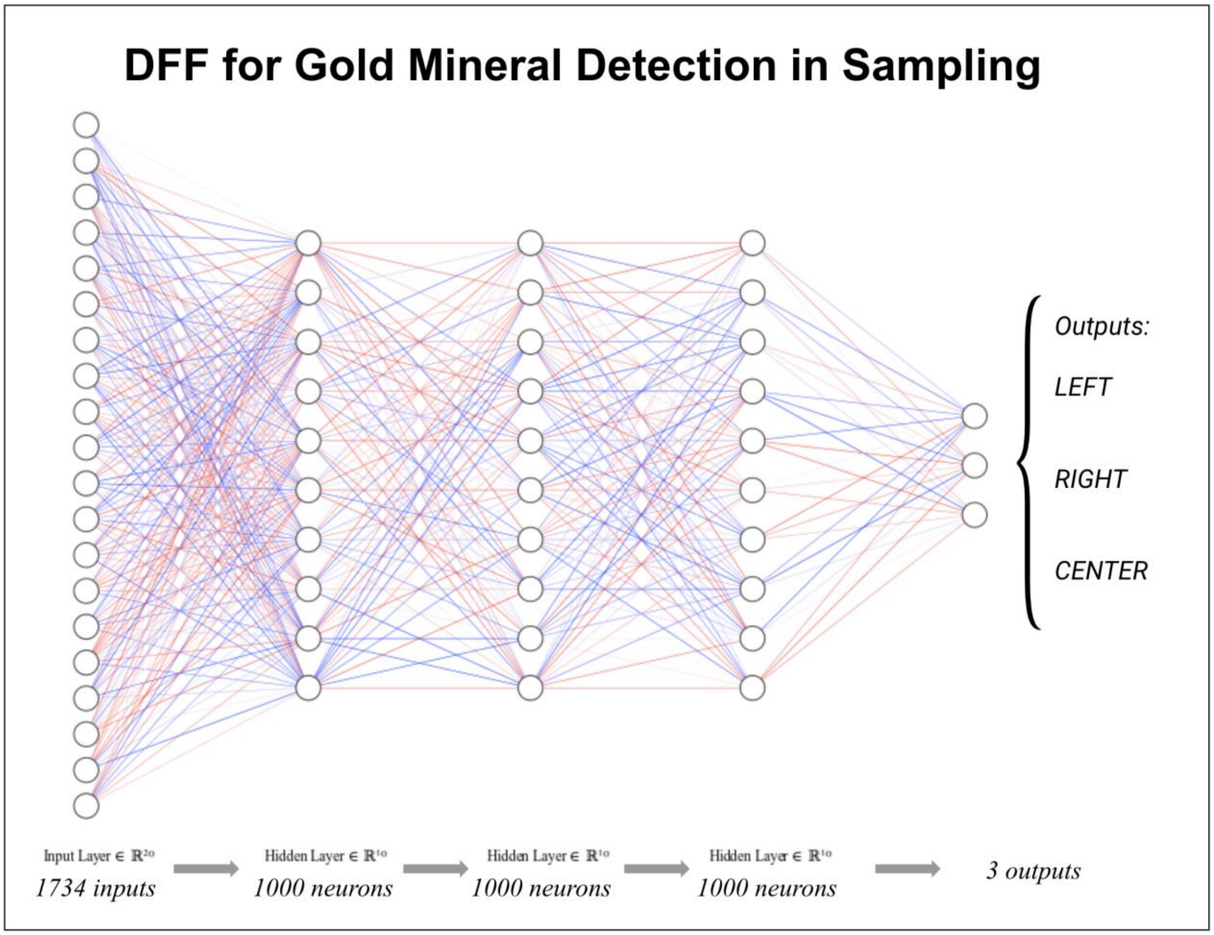
Creating An Artificial Neural Network (ANN)

Conceptualization and implementation: C110, C111, C112, C113, C114, C115

Before the integration of TensorFlow Lite into the official FTC App, we created a neural network with TensorFlow that could distinguish the left, center, or right position of the gold in the sampling minerals.

Choosing the Correct Structure

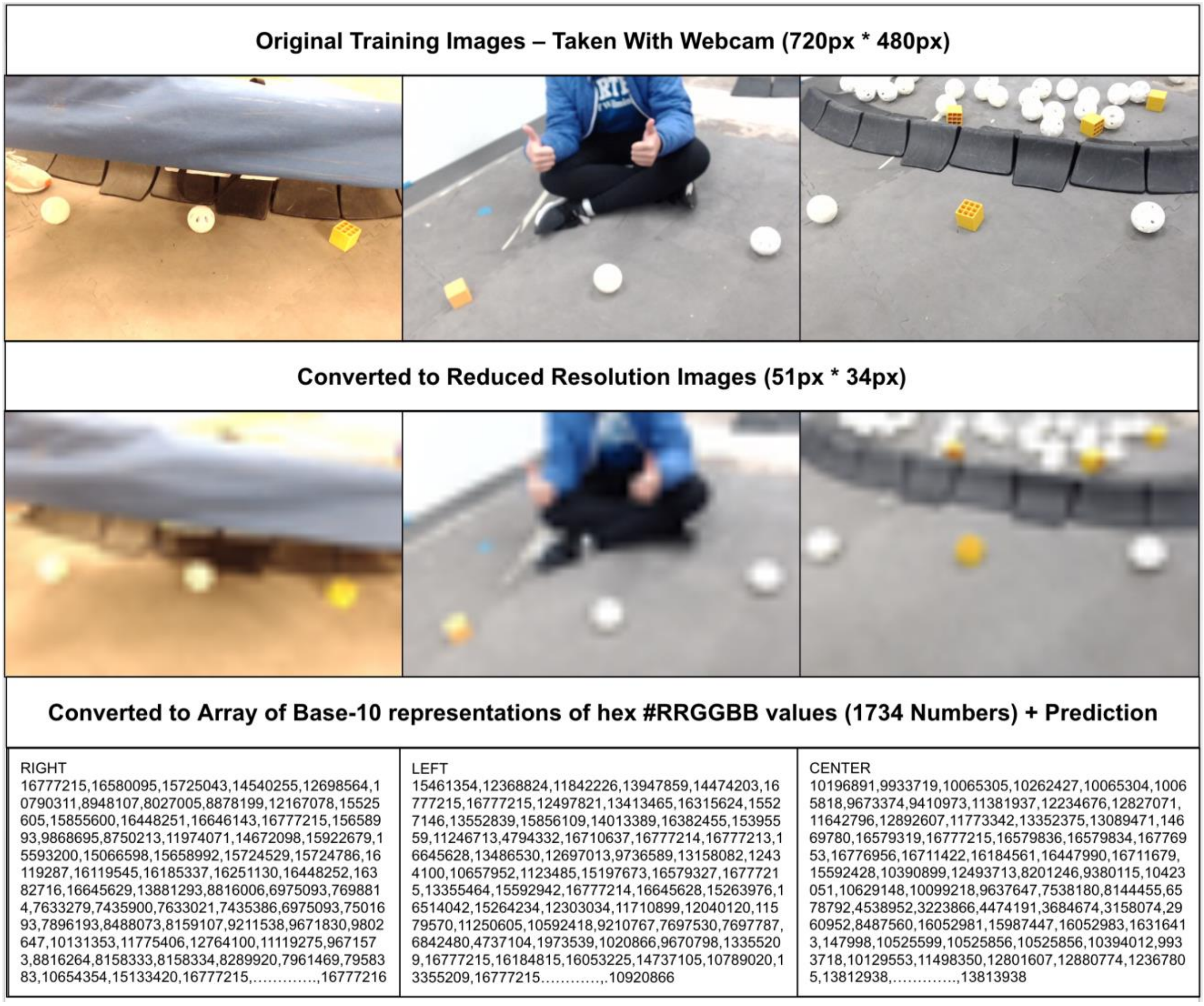
We decided to go with a ***deep feed-forward (DFF) neural network*** with ***backpropagation,*** a commonly used technique to train a neural network based around gradient descent.


DFF for Gold Mineral Detection in Sampling
Outputs:
LEFT
RIGHT
CENTER
Input Layer ∈ $\mathbb{R}^{20}$
1734 inputs
Hidden Layer ∈ $\mathbb{R}^{10}$
1000 neurons
Hidden Layer ∈ $\mathbb{R}^{10}$
1000 neurons
Hidden Layer ∈ $\mathbb{R}^{10}$
1000 neurons
3 outputs

There are 1734 inputs from the 1734 pixels in each of our input images, and 1000 neurons in each hidden layer. The final output has 3 possibilities. This neural network requires less training data because the problem at hand is fundamentally clear in terms of processing; there are no

complex edge detections required. All the neural network has to do is distinguish between yellow and white and their locations in the images.

Acquiring Training Data & Preprocessing (Total of 48 Images)



Original Training Images – Taken With Webcam (720px * 480px)

Converted to Reduced Resolution Images (51px * 34px)

Converted to Array of Base-10 representations of hex #RRGGBB values (1734 Numbers) + Prediction

```
RIGHT
16777215,16580095,15725043,14540255,12698564,1
0790311,8948107,8027005,8878199,12167078,15525
605,15855600,16448251,16646143,16777215,156589
93,9868695,8750213,11974071,14672098,15922679,1
5593200,15066598,15658992,15724529,15724786,16
119287,16119545,16185337,16251130,16448252,163
82716,16645629,13881293,8816006,6975093,769881
4,7633279,7435900,7633021,7435386,6975093,75016
93,7896193,8488073,8159107,9211538,9671830,9802
647,10131353,11775406,12764100,11119275,967157
3,8816264,8158333,8158334,8289920,7961469,79583
83,10654354,15133420,16777215,............,16777216
```

```
LEFT
15461354,12368824,11842226,13947859,14474203,16
777215,16777215,12497821,13413465,16315624,1552
7146,13552839,15856109,14013389,16382455,153955
59,11246713,4794332,16710637,16777214,16777213,1
6645628,13486530,12697013,9736589,13158082,1243
4100,10657952,1123485,15197673,16579327,1677721
5,13355464,15592942,16777214,16645628,15263976,1
6514042,15264234,12303034,11710899,12040120,115
79570,11250605,10592418,9210767,7697530,7697787,
6842480,4737104,1973539,1020866,9670798,1335520
9,16777215,16184815,16053225,14737105,10789020,1
3355209,16777215............,10920866
```

```
CENTER
10196891,9933719,10065305,10262427,10065304,1006
5818,9673374,9410973,11381937,12234676,12827071,
11642796,12892607,11773342,13352375,13089471,146
69780,16579319,16777215,16579836,16579834,167769
53,16776956,16711422,16184561,16447990,16711679,
15592428,10390899,12493713,8201246,9380115,10423
051,10629148,10099218,9637647,7538180,8144455,65
78792,4538952,3223866,4474191,3684674,3158074,29
60952,8487560,16052981,15987447,16052983,1631641
3,147998,10525599,10525856,10525856,10394012,993
3718,10129553,11498350,12801607,12880774,1236780
5,13812938,............,13813938
```

Key ideas in images for the neural network:

- Changed background of images to show background does not matter
- Changed lighting of images to show lighting does not matter
- Changed tilt of images to show tilt does not matter
- Did not change order of minerals to show **only** ordering of minerals matter

NOTE: The preprocessing was automated using Python scripts to save time.

The reason the images' resolutions were reduced was due to the fact that training the network would require more time. Although training the network at full resolution would be fine, it

would possibly take a few minutes, and this can get cumbersome when refining and tweaking the data. We felt predictions could be made just as well at reduced resolution.

The process of converting to a Base-10 representation of hex #RRGGBB values:

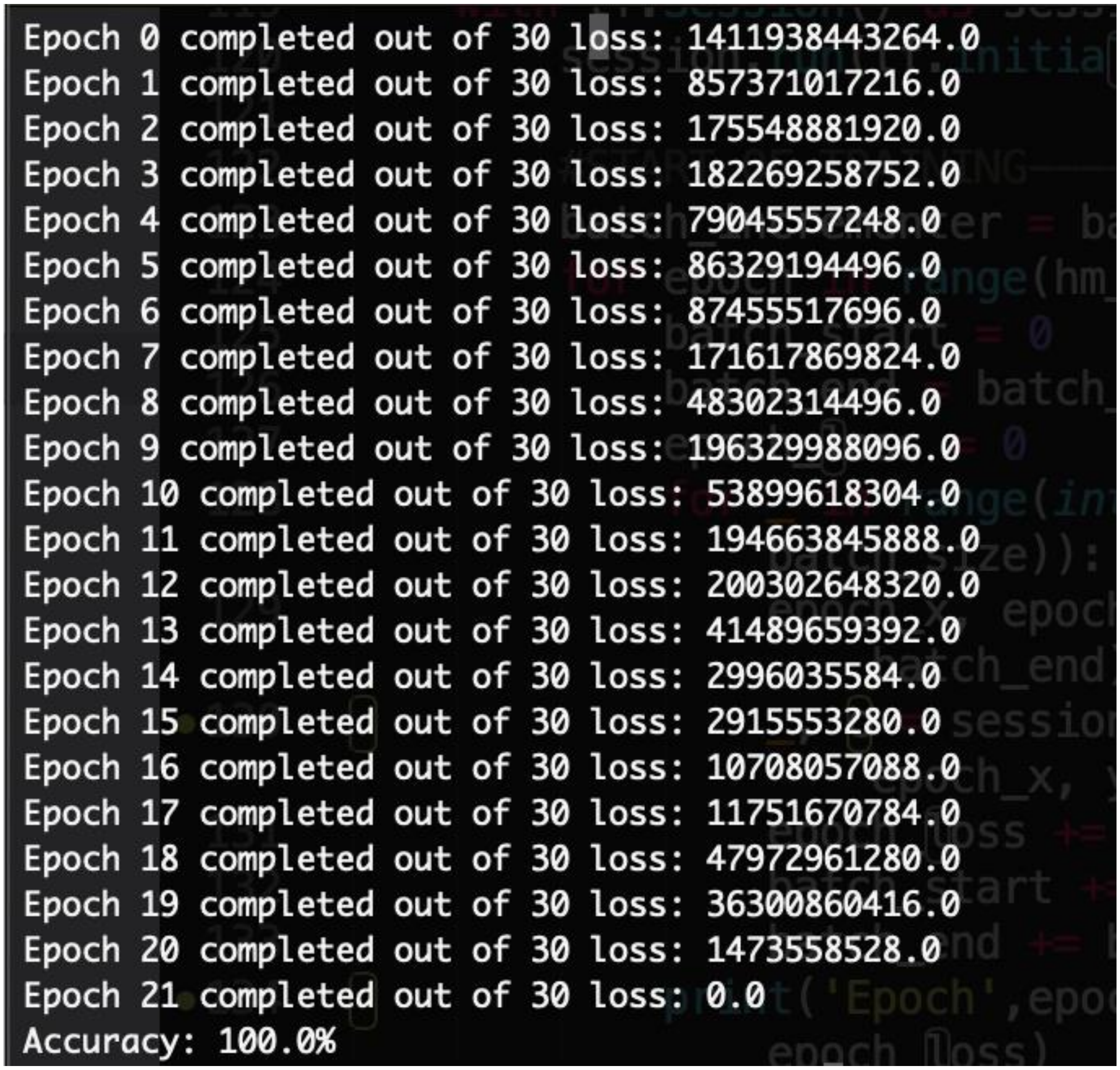
1. Take individual R, G, B (0 – 255) values of each pixel
2. Convert R, G, B values into one hexadecimal (base-16) number (#RRGGBB)
3. Take hexadecimal number #RRGGBB into a decimal (base-10) number

All the data was saved to a .txt file to be trained on later.

Training & Accuracy of Neural Network

Because we reduced the resolutions of the images in the preprocessing, training time for the 48 images was incredibly short: 5-15 seconds.

After training a successful model, here were our results.



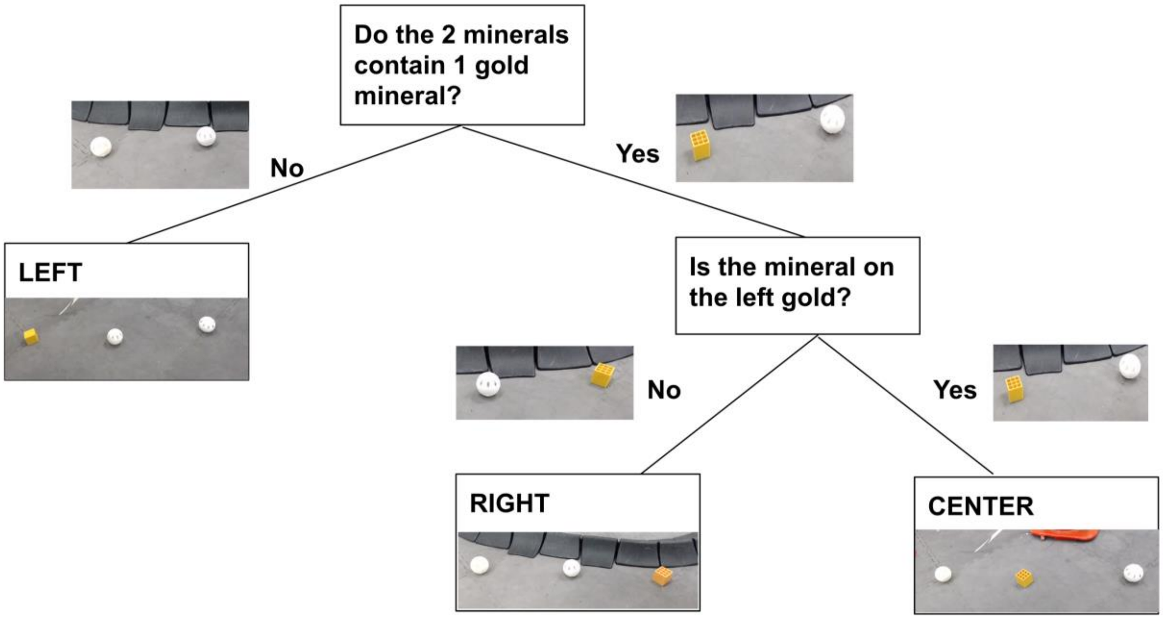
```
Epoch 0 completed out of 30 loss: 1411938443264.0
Epoch 1 completed out of 30 loss: 857371017216.0
Epoch 2 completed out of 30 loss: 175548881920.0
Epoch 3 completed out of 30 loss: 182269258752.0
Epoch 4 completed out of 30 loss: 79045557248.0
Epoch 5 completed out of 30 loss: 86329194496.0
Epoch 6 completed out of 30 loss: 87455517696.0
Epoch 7 completed out of 30 loss: 171617869824.0
Epoch 8 completed out of 30 loss: 48302314496.0
Epoch 9 completed out of 30 loss: 196329988096.0
Epoch 10 completed out of 30 loss: 53899618304.0
Epoch 11 completed out of 30 loss: 194663845888.0
Epoch 12 completed out of 30 loss: 200302648320.0
Epoch 13 completed out of 30 loss: 41489659392.0
Epoch 14 completed out of 30 loss: 2996035584.0
Epoch 15 completed out of 30 loss: 2915553280.0
Epoch 16 completed out of 30 loss: 10708057088.0
Epoch 17 completed out of 30 loss: 11751670784.0
Epoch 18 completed out of 30 loss: 47972961280.0
Epoch 19 completed out of 30 loss: 36300860416.0
Epoch 20 completed out of 30 loss: 1473558528.0
Epoch 21 completed out of 30 loss: 0.0
Accuracy: 100.0%
```

As shown on the image, the accuracy is 100% on our 48 images. The loss of 0.0 might be an indication of over-fitting the training data, but the network was able to successfully predict our test data, so the network indicates it has not overfitted to the point of inaccuracy. In other words, when we gave the network new data, it was able to successfully determine whether the gold mineral was in the left, right, or center.

Gold Mineral Decision Algorithm

See engineering notebook entries: C121, C122, C128, C129, C130

Since our robot was only able to see two minerals on the field, we had to write an algorithm to figure out which one is gold. To accomplish this, we guaranteed that the two minerals we saw would be the 2 right minerals out of the 3 in sampling. If the 2 were silver, the gold would be on the left. If the gold was the left of the 2 right minerals, it would be on the center. If the gold was on the right of the 2 right minerals, it would be on the right.



Do the 2 minerals contain 1 gold mineral?

No

Yes

LEFT

Is the mineral on the left gold?

No

Yes

RIGHT

CENTER

Vuforia Listener

(For the purpose of this, see *Realignment* in *A* Pathfinding Algorithm & Dijkstra's Algorithm*.)

To be able to retrieve the exact moment a Vuforia tag is seen by the camera, we took a unique approach in capturing these events. In Vuforia, there is a class known as the **VuforiaTrackableDefaultListener**. Normally, this class is called to check and retrieve a VuMark when one knows exactly when they will see a VuMark. However, due to the variable nature of our implementation of the A* and Dijkstra Pathfinding Algorithms (see *A* Pathfinding Algorithm & Dijkstra's Algorithm*), there is a need to know when a VuMark is found in a safe and efficient way.

This approach creates a class called **MOEListener** that extends the **VuforiaTrackableDefault Listener,** of which is used instead of the **VuforiaTrackableDefaultListener**. When overriding the methods in the default listener, we realized that there was no convenient method to realize when a *new* VuMark was found, so we modified one of the existing methods to let us know when a new VuMark was found. This way, in the event of a new VuMark, our robot would be notified properly.

Since Vuforia runs on a separate thread, there needed to be way to guarantee that the program would work consistently on multiple threads while being able to share information between the two. For this purpose, we utilized **Atomic** variables in Java for thread-safe sharing of information between the threads. If we did not, there is the slight chance that when two threads modify the same variable at the same time, there could be a loss of information. The usage of **Atomic** variables notifies the robot in the middle of following the path found by the Pathfinding Algorithms to stop what it is doing and realign against the VuMark.

Text-To-Speech (TTS)

For additional fun and utility, we incorporated the *Google Text-to-Speech* technology that allows text to be read aloud in a human-like fashion.

On the field, our robot likes to let us know how it is doing through a variety of phrases, including when it is initialized. Certain phrases include:

- “Initialized Vuforia”
- “Initialization Complete”
- “Initialized Gyro”

Over time, hearing these phrases can become cumbersome; however, our robot likes to spice things up. When completing certain tasks in autonomous, the robot likes to show its patriotism and loyalty to our team. Certain phrases it uses include:

- “Go MOE”
- “Whooo!”
- “Hi _____” (where _____ may be someone’s name)
    - Note: this is pre-programmed, we have not yet integrated the facial recognition technology for the robot to detect people on its own

Our robot also has a fondness for music, which it may play on or off the field. More than anything else, our robot’s personality makes interacting with it more interesting and fun! :)

MOE, Miracles of Engineering

FTC Team 365

2018-19 Control Award Submission

Introduction

Throughout the design of our robot, we have kept one universal theme in mind.

User Friendliness: The measure of how robust, simple, easy to maintain, and easy to use a robot is.

To accomplish this goal of user friendliness in Autonomous and TeleOp, we tried to keep the number of important components on the robot (sensors, motors, etc...) to a minimum while still vying to accomplishing our goals in mind. The result has been a robot that places a greater priority on intricate algorithms than sensors.

Our robot utilizes a good number of sensors, but wherever one can be omitted (for example: a camera rather than multiple color sensors), we take that option. This results in less environmental variables that can impact robot performance, as the robot relies on its algorithms and math to do computation in the place of sensors that could sometimes provide faulty data.

When driving the robot, we try to keep controls as simple as possible in order to allow the driver to focus on making important decisions rather than be distracted or bothered with controlling the robot.

Along with programming for the sake of the robot in competition, we have also programmed for the sake of learning (such as creating our own Neural Network!) to involve ourselves in other forms and kinds of programming. For an explanation on our thought process & more experimental procedures, view the Additional Summary Information. Also, below most titles will be a listing of notebook pages grouped together by what stage in the development process they show.

The 6 main sections are as follows:

1. Autonomous Objectives
2. Sensors Used
3. Key Algorithms
4. Driver Controlled Enhancements
5. Autonomous Program Diagrams
6. Additional Summary Information

Table of Contents

Autonomous Objectives

The following objectives are what we planned for in our robot's autonomous modes.

Autonomous Routine

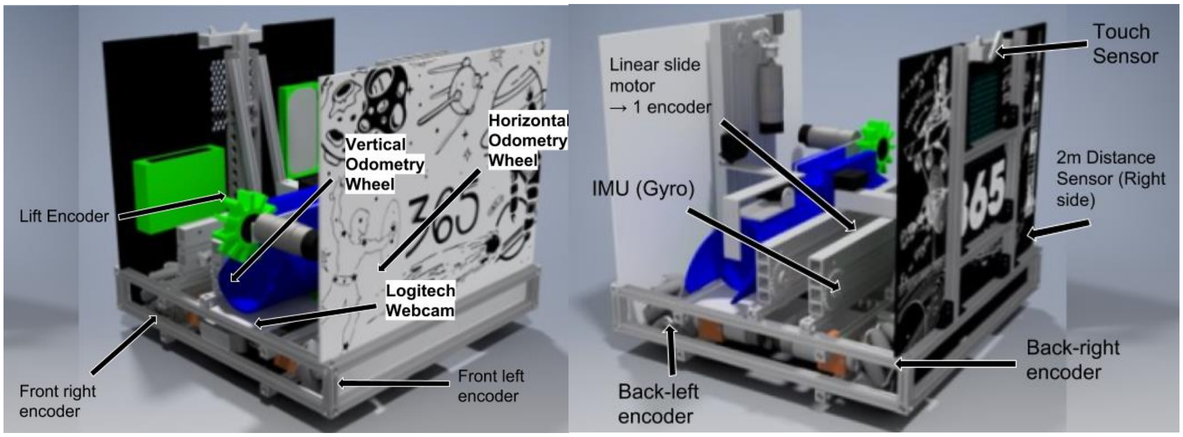
- Landing – dropping off of the lander (30 pts.)
- Sampling – knocking off the gold mineral (25 pts.)
- Sampled gold mineral placed in depot (2 pts.)
- Claiming – dropping the Team Marker in the depot (15 pts.)
- Parking – ending autonomous in the crater (10 pts.)

Algorithmic & Programming Objectives

- Establishing Field Grid & MOEPS (MOE Positioning System)
- Localization
- Multithreading
- Accurate Turning Methods
- Accurate Pathfinding with Intelligent Algorithms
    - Pathfinding Error Correction
    - "Rotational Symmetry"
    - Realignment
- Error Correction & Fallback Plan B Routines

Although not completely relevant in explaining the controls and actions of the robot, we included additional algorithms and details in our programming process that we felt to be of importance in the *Additional Summary Information*.

Sensors Used



Encoders

2: Encoders placed on motors involved with the robot’s mecanum drive. One encoder is on the front-left wheel, while the other is on the front-right wheel.

1: Encoder placed on the lift motor involved with dropping/hanging. The encoder is used for precise, controlled motion of the lift motor.

1: Encoder placed on the motor controlling the linear slide of our harvester. The encoder is used for making sure the slide does not swing out uncontrollably during autonomous, and allows for controlled motion during TeleOp.

Inertial Measurement Unit (IMU)

1: IMU build into the REV Expansion Hub. This IMU is effectively a gyro sensor, with the capability to measure rotation on 3 axes. We primarily use the horizontal axis, or the one that measures rotation parallel to the ground for accuracy in any turns or rotational movement of the robot.

Logitech Webcam

1: Logitech Webcam used in the front of the robot. Using this rather than a phone camera allows for the phone to be safely protected within the robot, making sure that nothing goes wrong during the match. The Logitech Webcam is used for recognizing the Vumarks.

REV 2m Distance Sensor

1: REV 2m Distance Sensor placed on the front side of the robot, facing the forward direction. The sensor is primarily used for telling distance away from the walls of the field.

1: REV 2m Distance Sensor placed on the right side of the robot, facing the right direction. The sensor is primarily used for telling distance away from the walls of the field.

Color Sensor

1: Color sensor placed on the front of the robot, facing the ground to align with lines during autonomous.

Touch Sensor

1: Touch Sensor placed near the top of the Tetrix channel used for holding the linear actuator used in dropping/hanging. The sensor is used for safe and automated resetting of the lift in hanging. Since the lift has to have the same starting point across all robot autonomous modes, a standard starting point is important.

Odometry Wheels

1: Vertical odometry wheel placed on the side of the robot to allow the robot to continually localize (know its absolute x,y position). An odometry wheel is an unpowered wheel that moves only when the robot moves on the floor. (See more in *Odometry + Gyro Localization*)

1: Horizontal odometry wheel placed on the side of the robot to allow the robot to continually localize (know its absolute x,y position). An odometry wheel is an unpowered wheel that moves only when the robot moves on the floor. (See more in *Odometry + Gyro Localization*)

.

Key Algorithms & Constructs

Field Grid & MOEPS (MOE Positioning System)

Conceptualization and implementation: C7, C8, C13, C14

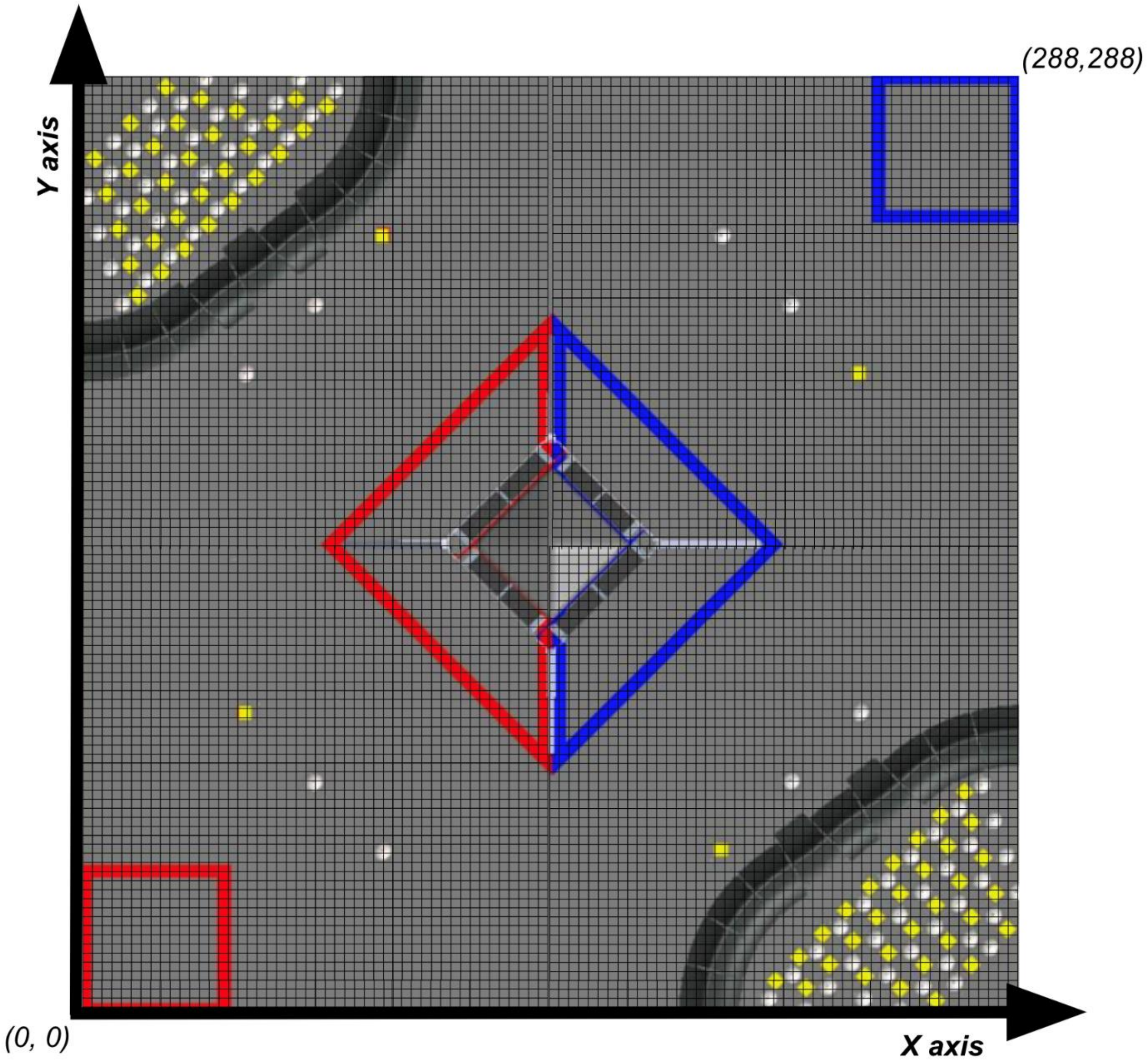
Due to the importance of the two positioning systems described below in our programming structures, we have affectionately coined the term MOE Position System, or MOEPS, to describe the systems.

Many of the following approaches and techniques we use rely upon a grid of (x, y) points. To form this grid, we divided up the field into a grid on the Cartesian plane, from (0, 0) to (72, 72).

A real field is 12ft. x 12ft., and we divided up the field into a 72 x 72 grid, where each MOE unit = 2 inches.

12 feet = 144 inches = 72 MOE units

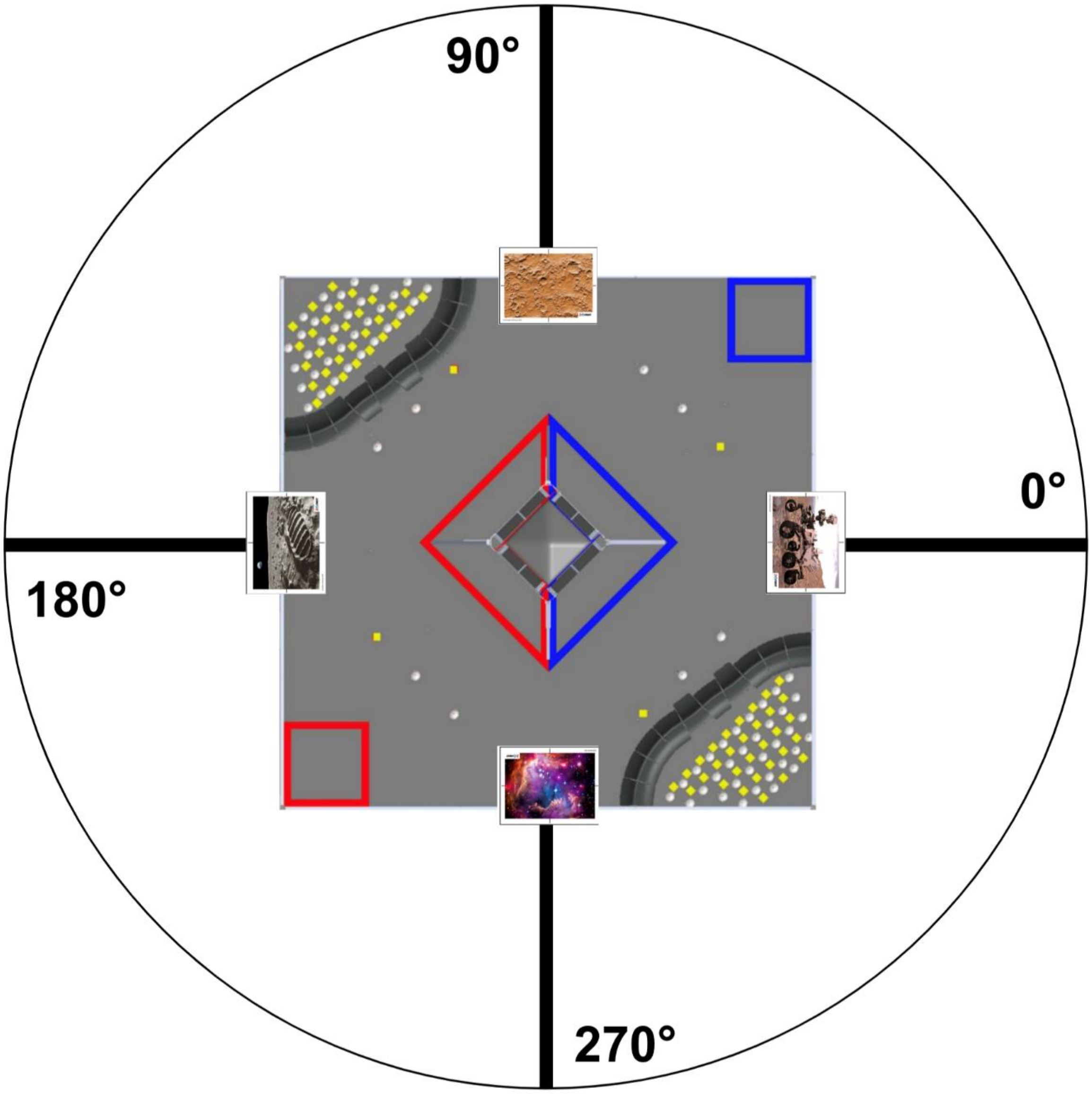
Additionally, the corner of the red depot was used as (0, 0) of the grid. Any of the four corners could have been used as (0, 0), so it was an arbitrary decision to choose the red depot.



To ease our programming, we made a Java class called PointMap to hold all important (x, y) points on the field with English names. While programming, we were able to refer to these names rather than the actual (x, y) coordinates. The following places were labelled as important:

Lander, Red Side Crater, Blue Side Crater, VuMarks, Field Corners, Sampling At Red Crater, Sampling At Blue Crater, Sampling At Red Depot, Sampling At Blue Depot, Red Depot, Blue Depot

Along with a positional (x, y) global map, we wanted to create an orientational global map to establish a consistent angle at any point on the map. The angle map was modeled off of the *Unit Circle*, which was used as a standard for marking angles on the map.

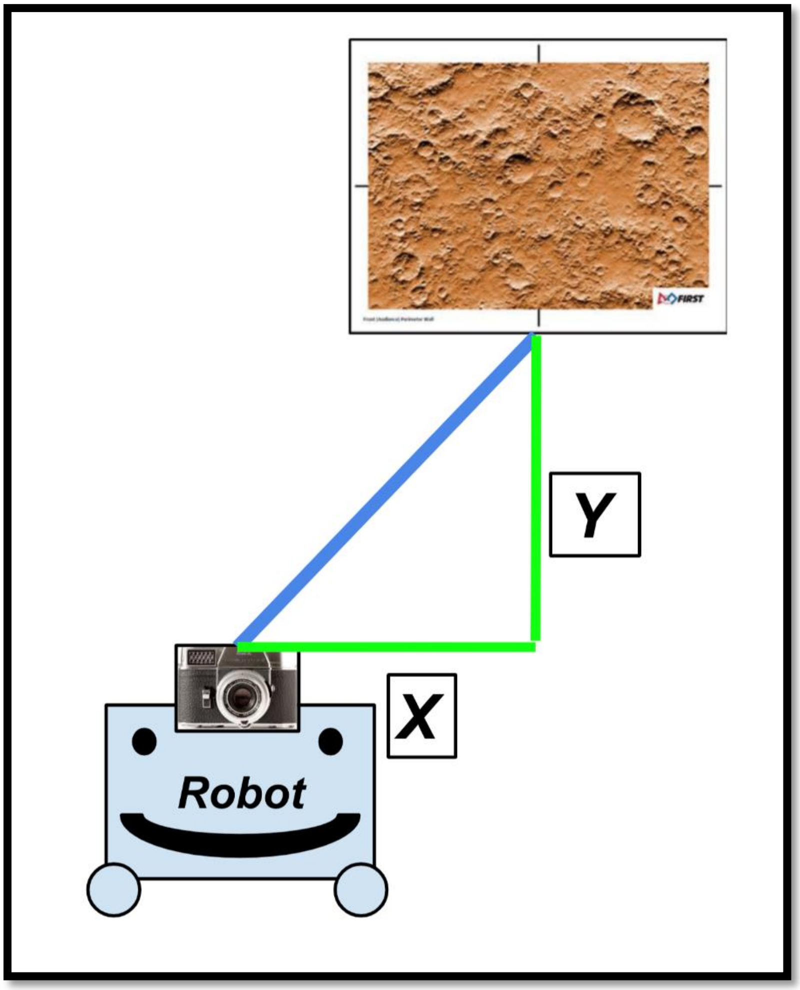

90°
0°
180°
270°

Localization

See engineering notebook entries: C42, C43, C146, C147, C148

In terms of our programming team, localization means finding out the robot's exact global (x, y) position on the field. In this case, the robot would have to find out its global (x, y) position on the **MOEPS global field grid** (check *Defining The Field Grid*). To accomplish this, we make use of the **Vuforia** image recognition technology and the **REV 2m Distance Sensor**.

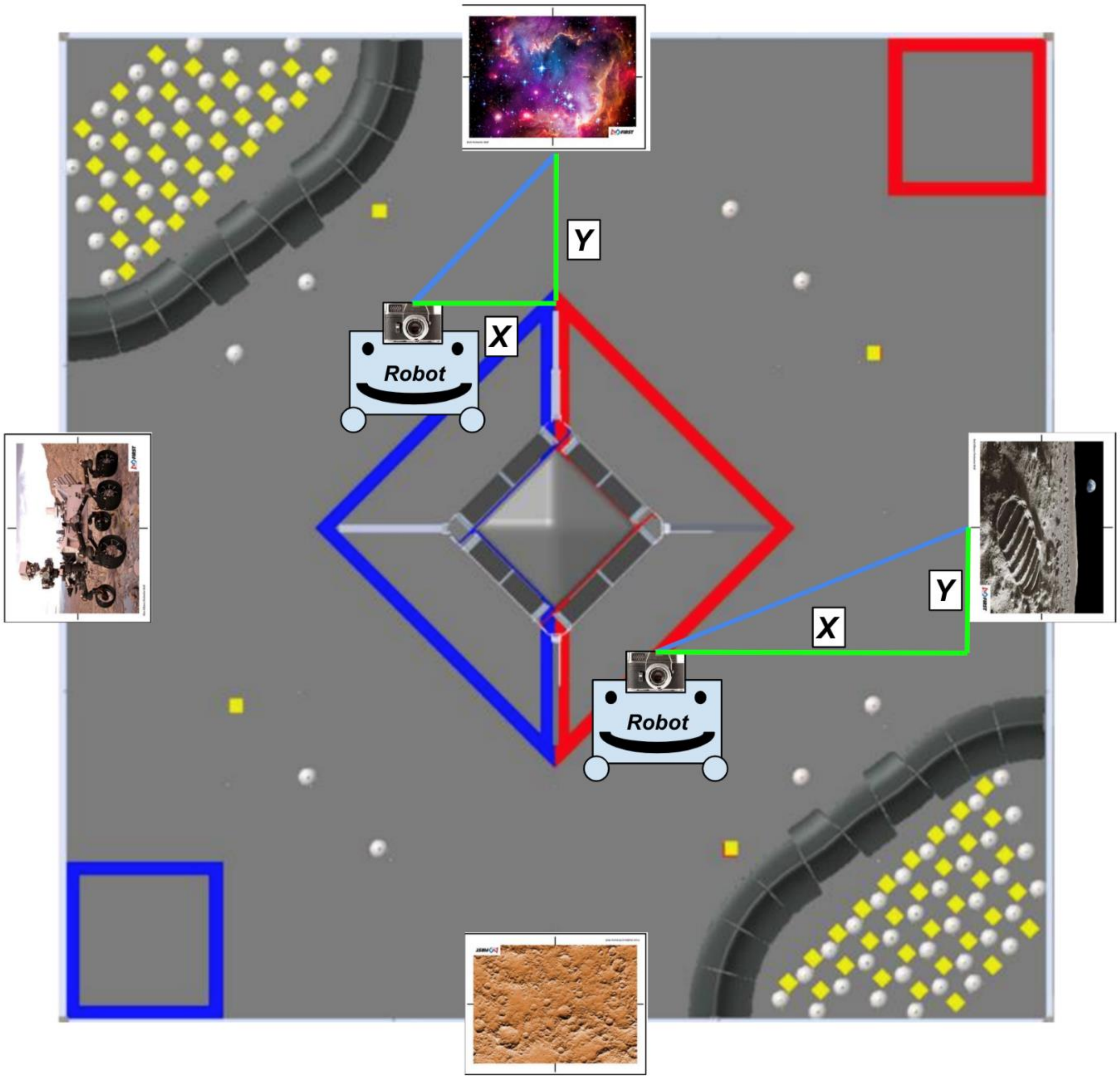
VuMark Localization



When the robot's webcam sees a VuMark, the following steps are taken:

1. Extrapolate horizontal (x) and vertical (y) distance from the VuMark using Vuforia
2. Scale the x, y distance into our 2-inch units – this is done by multiplying the values by the scalar 1/50
3. Depending on the VuMark, subtract the x, y values as appropriate – each VuMark has a distinct x, y position on the field, so subtract the robot's local x, y position from the VuMark from the VuMark's global position

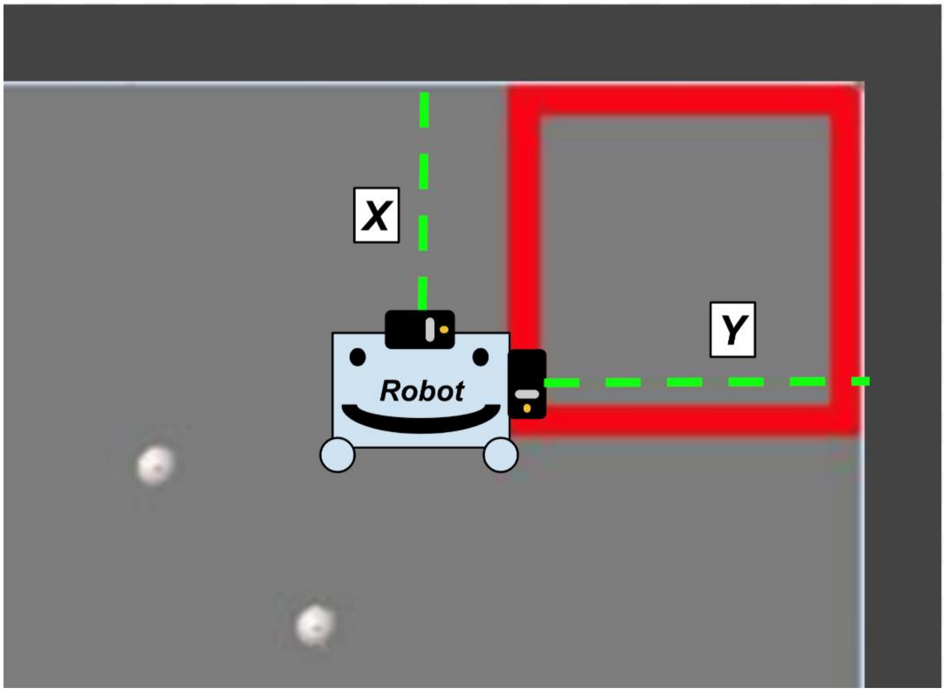
VuMark Localization on the field



Distance Sensor Localization

Localization using the distance sensor is similar to the VuMark Localization method, but is less reliable due to inaccuracies and errors that occasionally occur in the distance sensors. It used when no VuMarks are available and localization is necessary.

In this case, the following steps are taken: 1. Get vertical and horizontal distance in inches from wall with distance sensors 2.



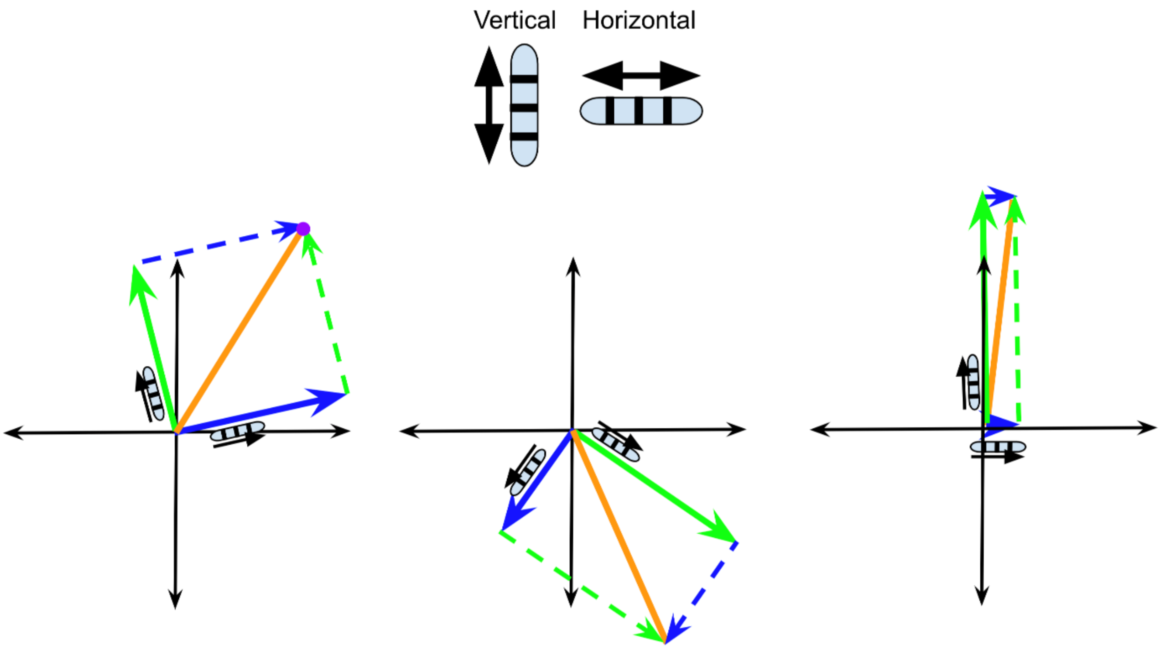
Subtract inches from wall (x, y) position
3. Resulting (x, y) coordinate is the robot's global location

Odometry + Gyro Localization

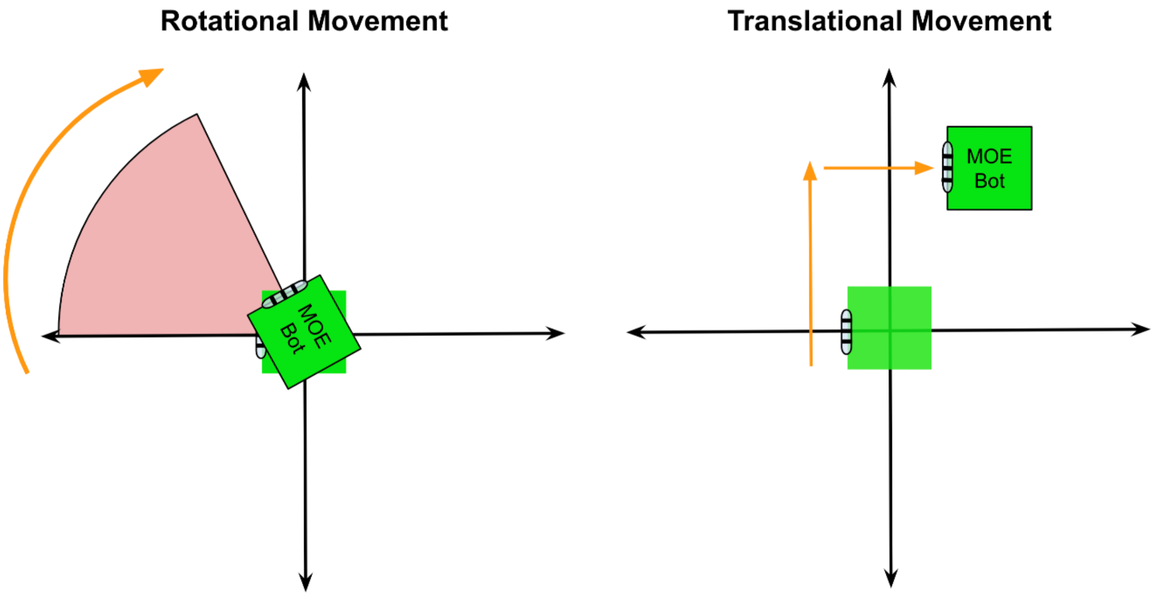
Although the above options for localization *(Vuforia, Distance Sensor)* are robust, they both contain one fatal flaw: they are not universally accessible. To use the above options, the robot either has to be close to a VuMark or has to be close to a corner on the field while also being in the correct orientation.

Then came about the idea of using Odometry Wheels, which allow for continual localization at any point on the field.

Odometry wheels are non-powered wheels bound to an encoder that allow for accurate measurement of robot's absolute movement. Since these wheels are not part of the drive train, when the robot drives forward into a wall, for example, the drive train motors would register a change in encoder tics while the odometry wheels would remain consistent since they only move with the robot. In our robot, we use 2 MA3 encoders and 2 omni wheels for a total of 2 odometry wheels.



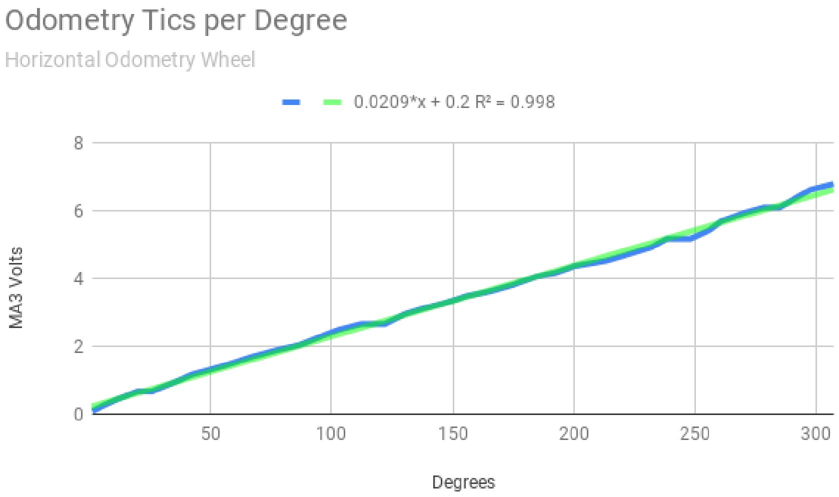
The two odometry wheels on our robot allow for accurate measurement of positional movement. Through the usage of the gyro sensor, we are able to do rotational math to figure out the robot's position at all times.

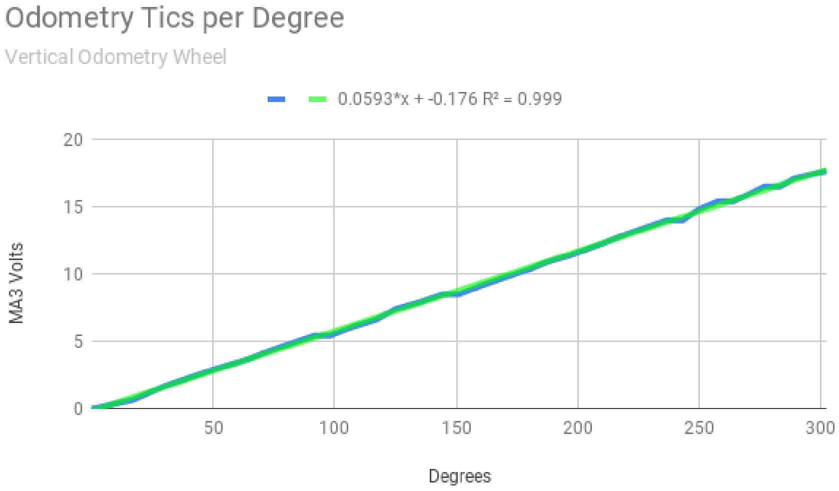

Rotational Movement

Translational Movement

To use the odometry wheels reliably, we have to be able to distinguish between rotational movement and translational movement. The typical way to solve this issue is through a mechanical paradigm, which involves the use of additional odometry wheels opposite to each on of importance. By averaging the values of both wheels (positive & negative changes would cancel out), one could get an accurate measurement of only translational movement. Applying this methodology to our robot, we would have 4 odometry wheels on the robot. Due to the fact that we did not have space for 4 odometry wheels, our team has to resort to other options to discount rotational movement.

This has been done through the gyro and a measure we created known as “rotational offset”. This “rotational offset” would be used as a subtractor to discount any non-important odometry values. By dividing the angle difference (found between each refresh of the odometry wheels’ position) over 360, and multiplying by the “rotational offset”, an appropriate offset measure can be found for each wheel.





By calibrating & turning a “rotational offset” for each of the two odometry wheels, we can accurately discount any rotational motion, allowing only translational movement to be used in the calculation of the robot’s position.

The above graphs show the number of volts that the MA3 encoders showed for a rotation of the robot. By using the equations above to estimate the rotational offset for each of the wheels, we can effectively & accurately cancel out rotational motion.

Multithreading

Implementation of lift mechanism: C140

Implementation of realignment: C58, C59

Multithreading is a technique by which a single set of code can be used by several processors at different stages of execution. In other words, a program can have multiple sets of instructions running at the same time. With multithreading, the robot is able to do *more than one task* at any given time. Since our robot is trying to accomplish all standard autonomous points, time is often cut close to 30 seconds.

Without the use of multithreading, the robot's autonomous routine would have to speed up its motors significantly to meet the allotted 30 seconds. This speeding up results in less accuracy, resulting in an autonomous that is more prone to error.

Although processes like Vuforia and TensorFlow may run on separate threads, we intentionally use our own threads or pull from other threads for the following purposes:

- Bringing down the lift mechanism used for dropping/hanging
- Pathing algorithm realignment (see *Vuforia Listener* in *Additional Summary Information* for more details)

We also used **Atomic variables** for thread-safe operation. When a global variable is dealt with between 2 or more threads, there is always the danger of it leaking data when operations on it are done at the same time. Since using a raw variable without synchronization or any other standard is considered bad practice, we decided to use Atomic variables for thread-safe operation. This way, when communicating between the Main Robot thread and the Vuforia thread, we can guarantee that no strange behavior in autonomous occurs because of loss of data between the two threads.

Turning Methods

Conceptualization and implementation: C31, C32

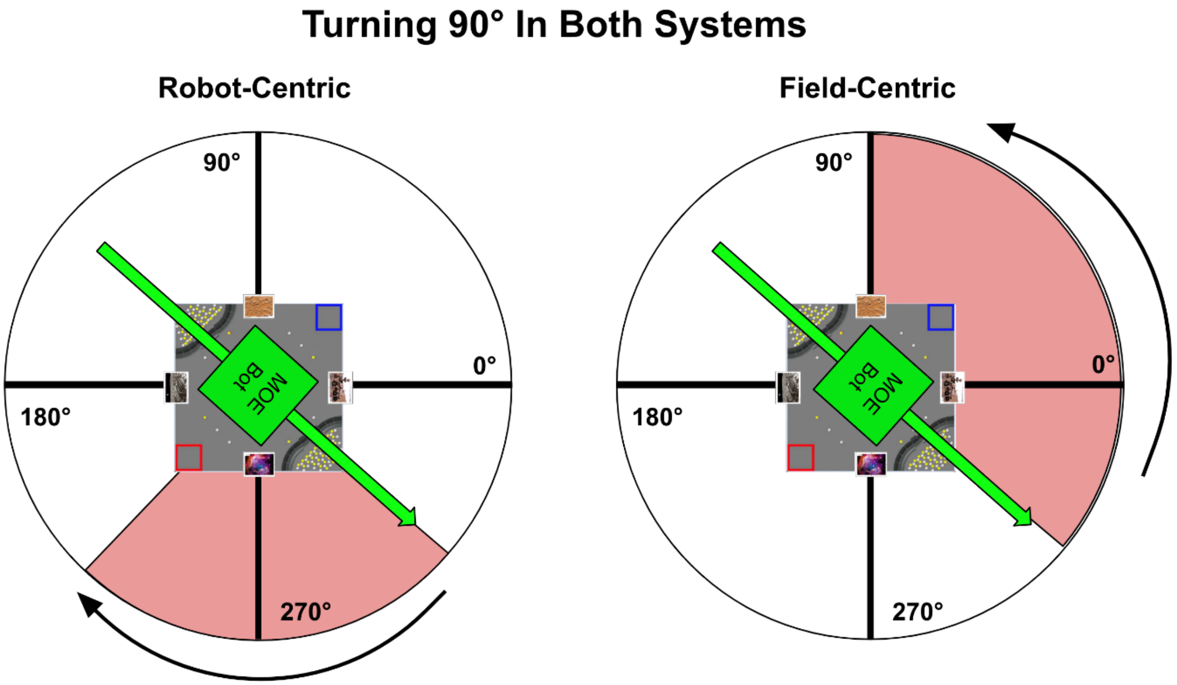
Turning in autonomous must be precise to the degree for repeatable results, which is why turning is dictated by the **IMU** sensor built into the ***REV Expansion Hubs***. Instead of turning by time, we turn by setting the powers the motors, and simply wait for the **IMU** to indicate that we are within the correct angle.

Field-Centric Turning & Robot-Centric Turning

In **field-centric turning,** the **MOEPS angle map** described above (see *Field Grid & MOEPS*), the robot turns to a given global angle on the field.

In **robot-centric turning** the robot turns to a given angle relative to itself.

Robot-centric vs. Field-centric Turning:



As shown in the diagram above, the robot turns 90° directly to the right in the robot-centric turn, while the robot is turning to the 90° mark in the field-centric turn. No matter the orientation, the robot will always turn to the same 90° mark in field-centric turning.

Jump Point Search / A* / Dijkstra's Pathfinding Algorithm

*Conceptualization and implementation of old linear pathfinding algorithm (**not used on robot**):* C42, C43

Conceptualization and implementation: C46, C47

Implementation and testing: C50, C51

Debugging and gradual improvements: C55, C56, C58, C59, C64, C76, C93

Radius and size reductions: C97, C98, C99

2nd Stage Debugging: C104, C105

8-Directional Movement: C128, C129

3rd Stage Testing: C146

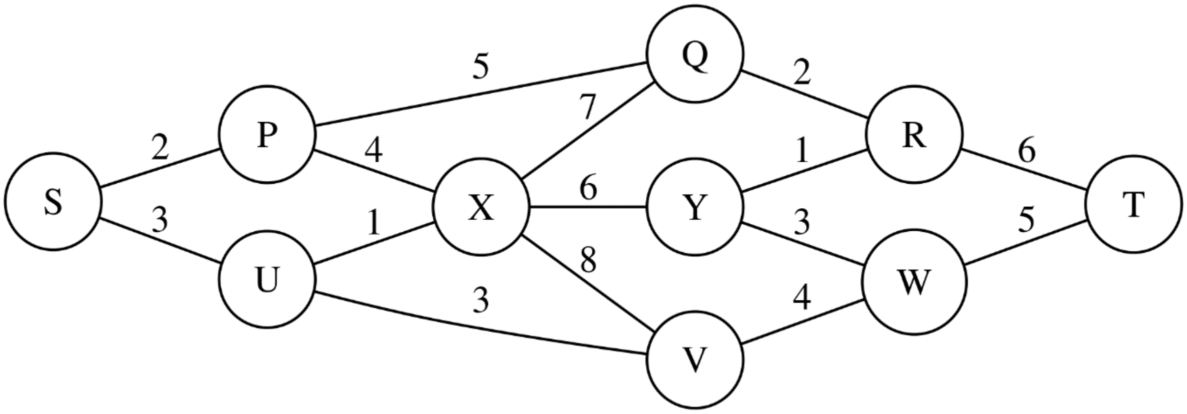
"Rotational Symmetry": C150, C151

Introduction

The purpose of the algorithm is to allow the robot to dynamically figure out how to reach its destination. In many autonomous pathings, whenever there is a slight disturbance, the autonomous fails to finish. Rather than explicitly giving the robot a path to follow, the robot is given an end destination. Through localization, the robot figures out its (x, y) coordinate on the field and calculates on its own how to reach the destination.

The *A* (pronounced A Star)* and *Jump Point Search Algorithms* are similar to the popular *Dijkstra's Algorithm*, which is used for finding the shortest paths between nodes in a graph. The primary difference between Dijkstra's and the other two is that the pair utilize a "heuristic function", or an approximation function, to approximate a faster solution to Dijkstra's algorithm. Dijkstra's algorithm checks many more cases than the A* Algorithm, therefore taking longer to arrive at a similar answer. Since the field we are using is 288x288 (82944) nodes, we wanted to guarantee that processing speed would be fast. The algorithms commonly deal with graphs shown like the one below, but had to be specially adapted in our case to work with a 2D grid.

Visual representation of traditional graph in computer science:



To account for processing speed & time, Dijkstra's Algorithm has a worse case time complexity (when using lists) of **O(N^2)** where N = number of nodes on the graph, while A* *generally* has a time complexity of **O(b^d),** where b = branching factor and d = depth of the solution on the search tree. However, both of these algorithms have a very slow runtime in certain circumstances, taking over 20 seconds to run. This is unacceptable when run in autonomous, which only has a period of 30 seconds. The Jump Point Search algorithm is an optimized version of the A* pathfinding algorithm that consistently brought our runtime below 2 seconds.

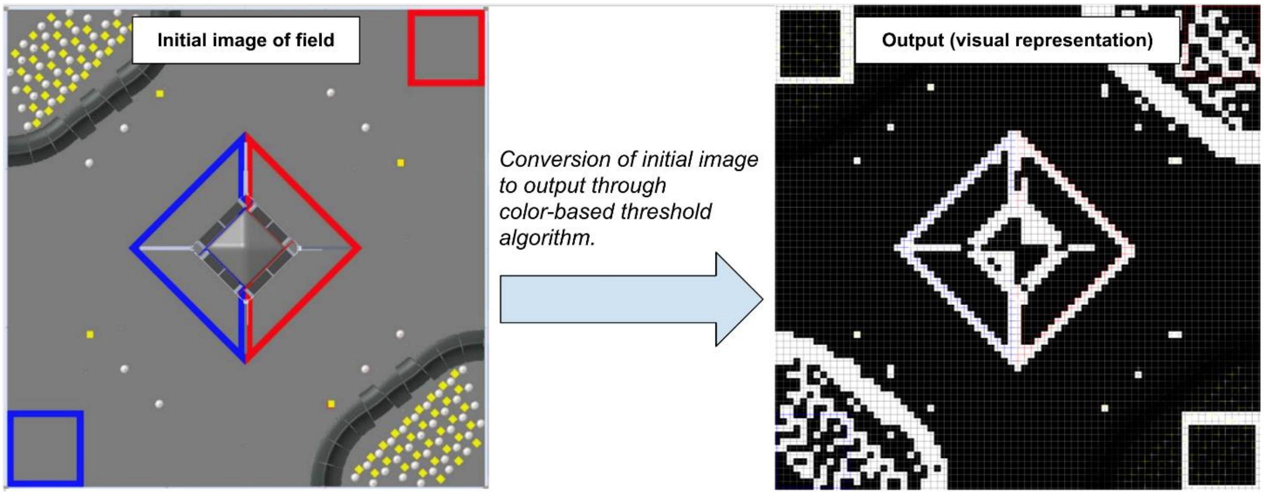
Note that the time complexity of A*/Jump Point Search is worse when using a very expensive heuristic cost function, but we are using the simple Euclidean distance, or the distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

With the pathfinding algorithms, the robot is able to move in **8 directions** to go from point A to point B. The directions are labelled as follows: **North, Northeast, East, Southeast, South, Southwest, West, and Northwest.**

Setup

To utilize the **pathfinding algorithms**, we needed to first setup a graph. To accomplish this, we took a 2D image of the field from Game Manual 2. After that, we wrote a **Python** script (utilizing the *PIL imaging library*) to go through the image, converting it to points we deemed as barriers (white) and points we deemed as free space the robot could travel on (black). This conversion was done through a color-based threshold. In essence, the gray parts of the map were free space while the other colors were barriers. The output was an image with the converted points as well as a 288x288-*dimensional array* that we would be able to use as our graph for the pathfinding algorithms. Also, the image was flipped because we wanted [0,0] of the 2D array to be the corner of the red depot, and [287,287] of the 2D array to be the corner of the blue depot.

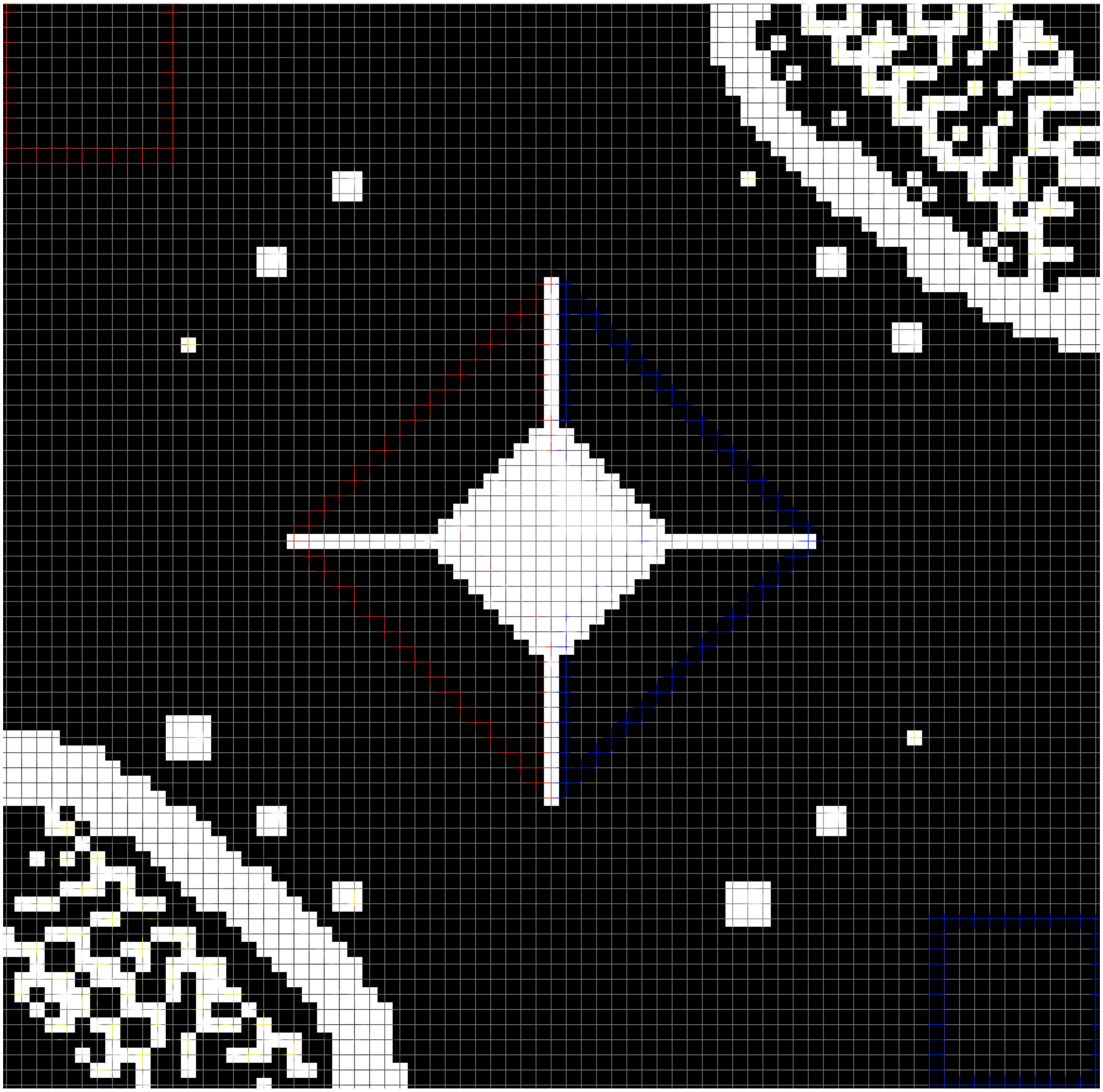


Conversion of initial image to output through color-based threshold algorithm.

Mapped FTC Field (Visual Representation of Array)

1 (or white) = point a robot cannot travel on

0 (or black) = point a robot can travel on



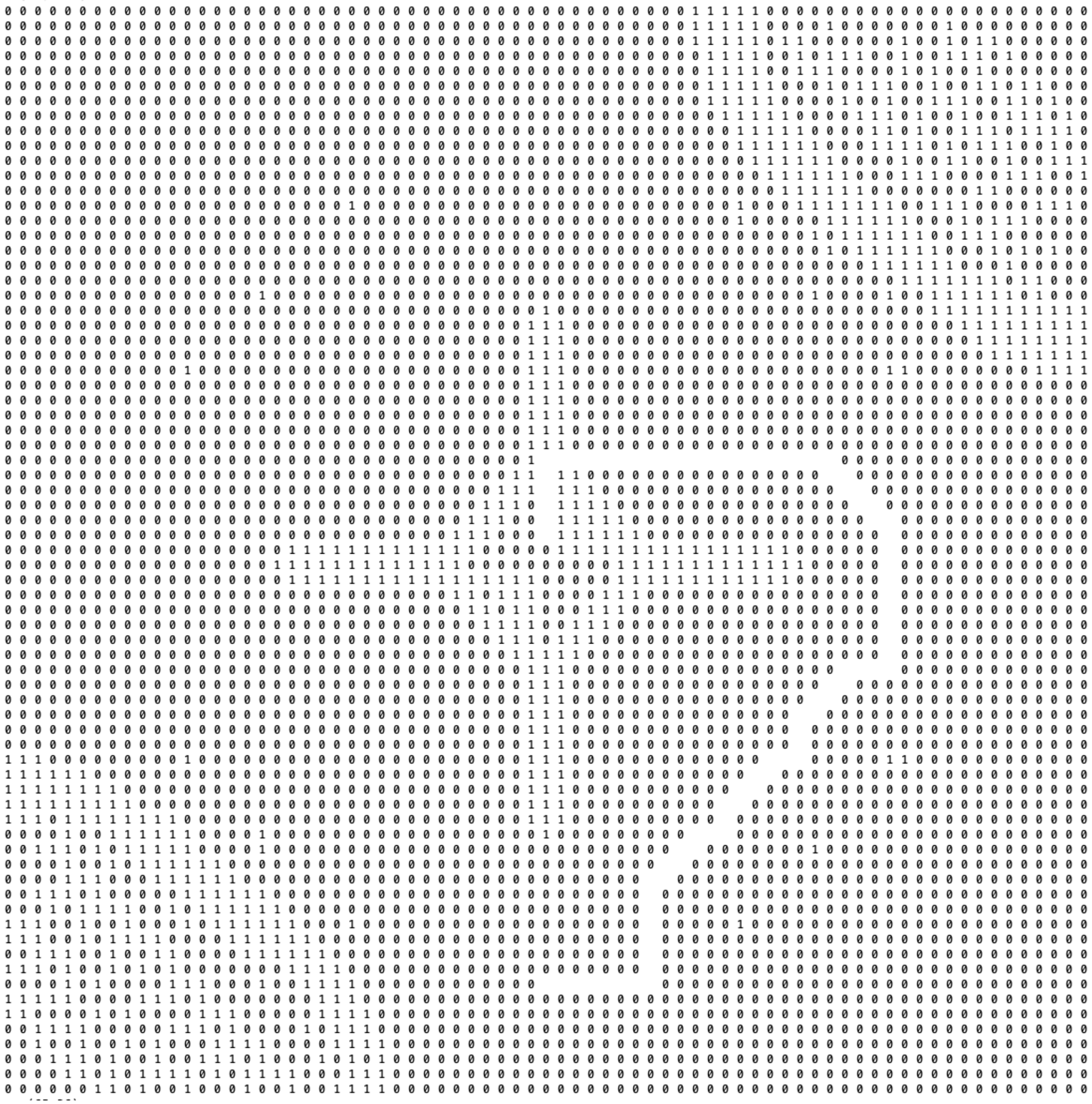
The original conversion had some errors, because places (depot, lines near the lander, etc...) were marked in white when they should have been open space. To fix this, we manually changed some values in the array. Since most of the conversion work was done by the Python script, this only took a few minutes.

The above image is what the output array looked like visually. The barriers shown in the image above are represented by 1s, while the free space shown in the image above are represented by 0s.

Implementation

We implemented the algorithm in Java, making a separate class to handle the calculations. To verify that we wrote the algorithm correctly and be able to make predictions on robot movements, we made a simulation to show the pathfinding algorithm's path from any Point A to Point B visually:

Initial Simulation

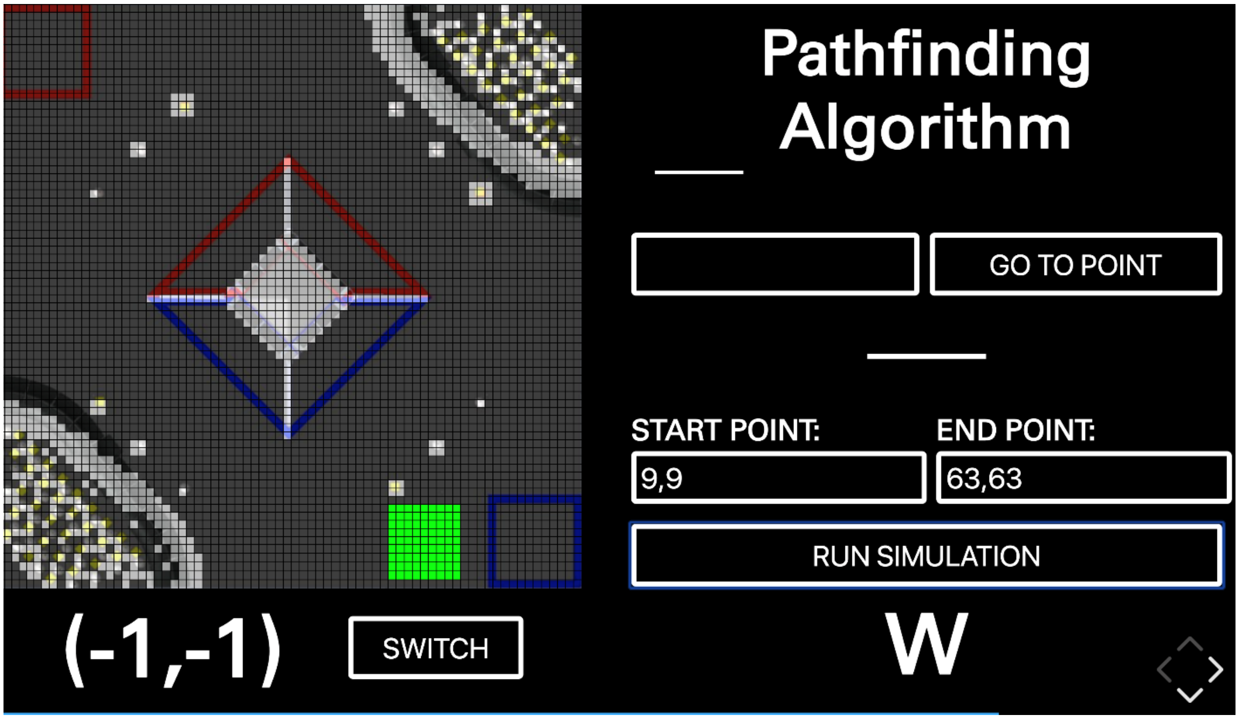


```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1                                         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1   1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1   1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0   1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0   1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0   1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0         0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0       0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0                 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

As complications with the algorithm increased, there was a need for a better simulation that more accurately depicted the algorithm in action. Below is a screenshot of an active simulation that shows the the robot (the green square), step by step, moving through the field.

Screenshot of Final Simulation

Video of live simulation: https://1drv.ms/v/s!AqOPfHs4_986ihlsFJLdiAYHtxG2



START POINT: 9,9

END POINT: 63,63

To now use the algorithm in practice, we had to convert the results into a usable format by writing an algorithm to do so.

Path Conversion Algorithm

(Input) - Original Pathfinding Results: A series of points describing each point to go from point A to point B. *For example, getting from (0,0) to (5,5) could be:* (0,0) --> (0,1) --> (1,1) --> (2, 1) --> (3, 1) --> (3, 0) --> (4, 0) --> (4, 1) --> (5, 1) --> (5, 2) --> (5, 3) --> (5, 4) --> (5, 5)	**(Output) - Usable Results:** The number of inches in each direction the robot has to go, in order (each unit is 2 inches). *For example, getting from (0,0) to (5,5) could be:* FORWARD 2 in. --> RIGHT 6 in. --> BACKWARD 2 in. --> RIGHT 2 in. --> FORWARD 2 in. --> RIGHT 2 in. --> FORWARD 8 in.

The robot first turns towards the **90° mark** described in the **MOEPS global angle map (**the direction facing the Crater/Mars VuMark). The results from the pathfinding algorithm would then be translated into movements for the robot based on encoder ticks. The result of this extensive process is a robust and repeatable movement system that allows the robot to figure out its own path when given two points on the field. This simplifies the process of adjusting and programming autonomous, as well as allowing for a more robust and dynamic movement system.

Pathfinding Algorithm Error Correction

The pathfinding algorithm worked perfectly well in theory, but in practice, there were a few issues we had to fix in order of importance.

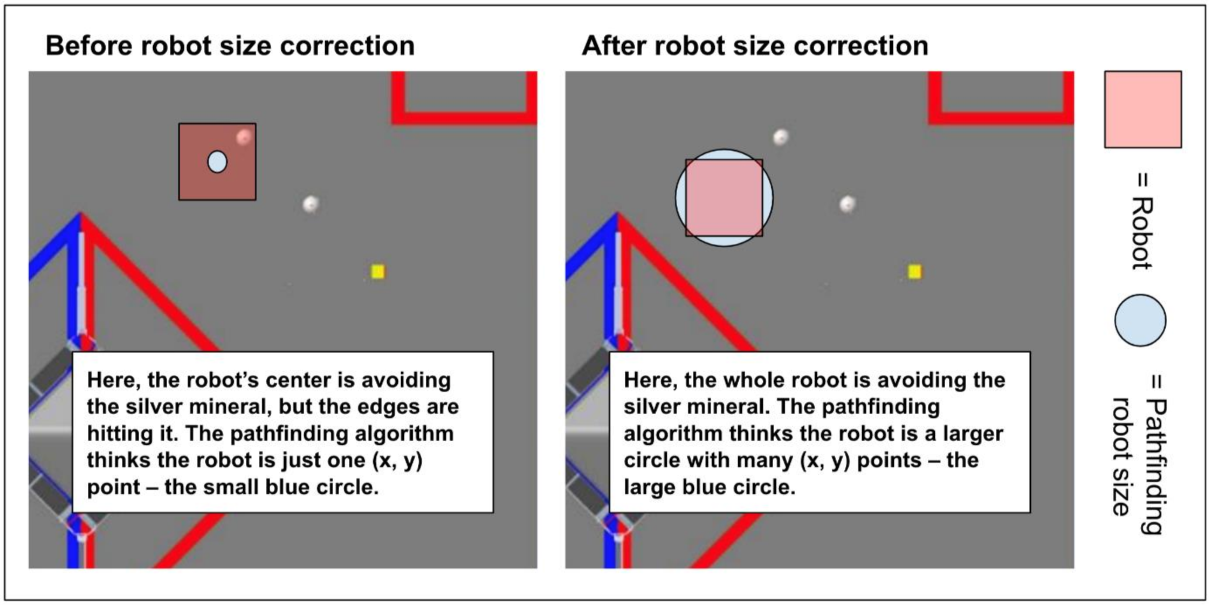
1. The **A* algorithm** treated the robot as a single (x, y) point on the **global 72x72 grid** while the robot actually comprised at least a circle of many points with radius about 10 inches. This led to the edges of the robot crashing into parts of the field (lander, crater, sampling, etc...) while its center thought it was following the pathfinding algorithms as a single small point.
2. While moving, the robot would turn slightly off angle. It would be not exactly at its end destination due to slight turning while making up, down, left, and right movements.

Error #1 – Size Corrections

Conceptualization and implementation: C55, C97

Second iteration: C98, C99

To fix this error, we modified the size of the robot in the algorithms. Instead of treating the robot as a single point, we treated it as a collection of multiple points – when put together, these points would form the robot rather than one small point.



Before robot size correction

Here, the robot's center is avoiding the silver mineral, but the edges are hitting it. The pathfinding algorithm thinks the robot is just one (x, y) point – the small blue circle.

After robot size correction

Here, the whole robot is avoiding the silver mineral. The pathfinding algorithm thinks the robot is a larger circle with many (x, y) points – the large blue circle.

Error #2 – Turn Corrections

To fix this error, we took the IMU sensor's horizontal angle before the robot followed the A* algorithm. We then constantly tracked the gyro sensor's angle while the robot followed the A* algorithm.

If the IMU's angle strayed by more than 2°, the robot self-corrected itself back to the correct angle by again utilizing the gyro to turn back into position.

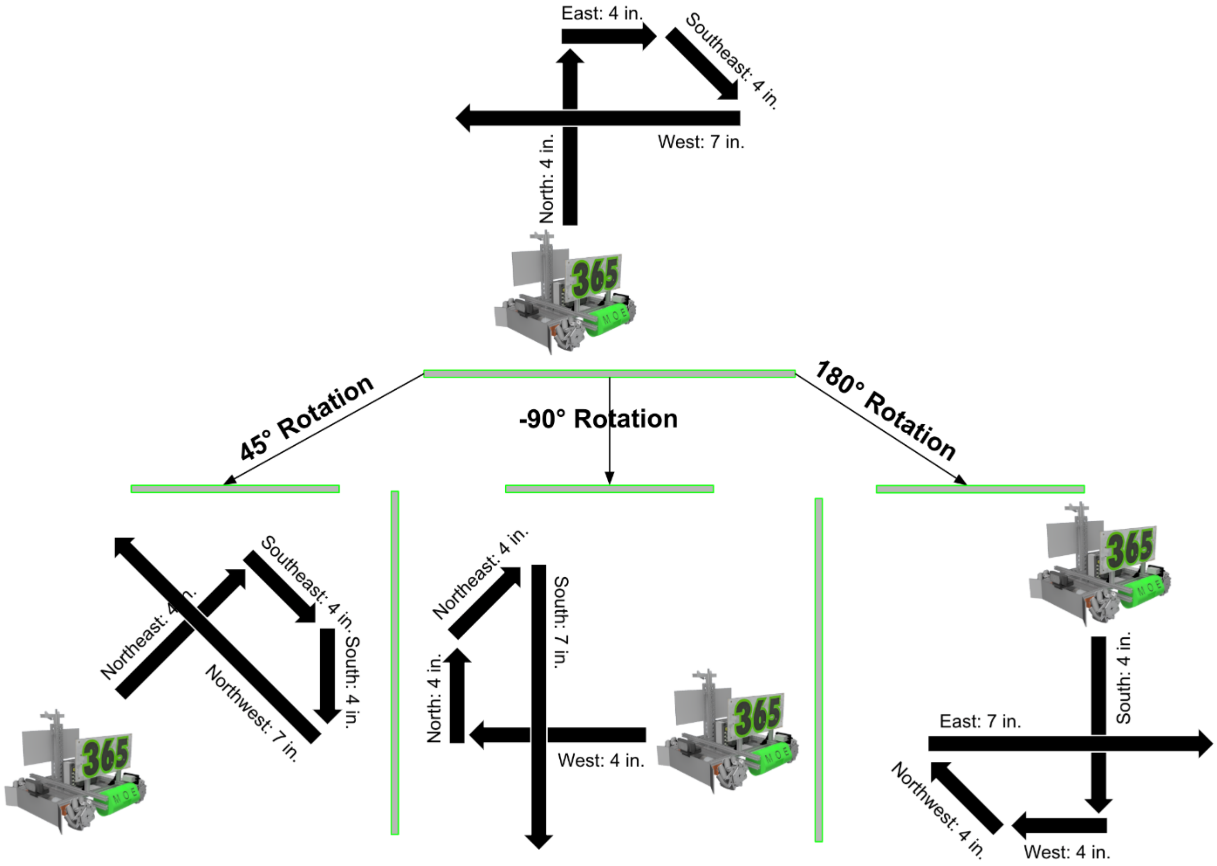
"Rotational Symmetry"

Conceptualization and Implementation: C150, C151

Another feature that we added to the pathfinding algorithm is the idea of “rotational symmetry”. In other words, a given set of output instructions can be rotated by certain number of degrees while still preserving the relative directions of each movement.

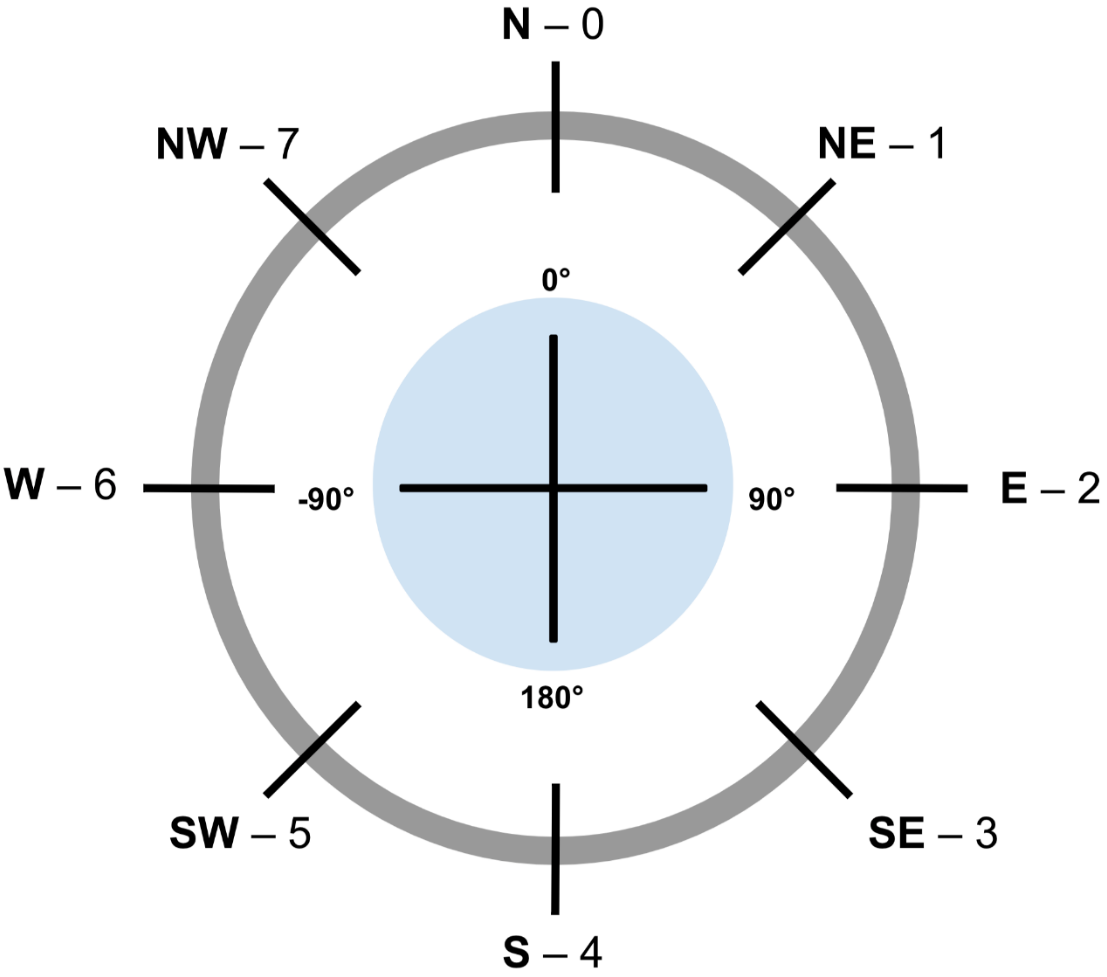
The purpose of rotational symmetry is to allow for optimizations in accuracy and speed of robot movements. Since moving forwards and backwards is always faster than strafing, rotational symmetry allows the robot to take a pathing from the Pathfinding Algorithms and rotate the pathing instructions. The robot can then quickly turn and apply the rotated instructions to allow for more forwards and backwards movements, resulting in a more robust movement.

Examples of rotations done on set of output instructions:



The algorithm is accomplished by setting a numerical value to each of the directions in clockwise order. North all the way around to Northwest is numbered from 0 up to 7. This simple numbering pattern makes rotation much simpler than writing each direction’s rotation to its right. To rotate a direction, divide the angle needed to rotate by 45° and add it to the number. In the case a value goes above 7, it is wrapped back around to start at 0 (ie: 9 becomes 2).

Visualization of Direction to Number mapping, along with degrees associated with rotations:



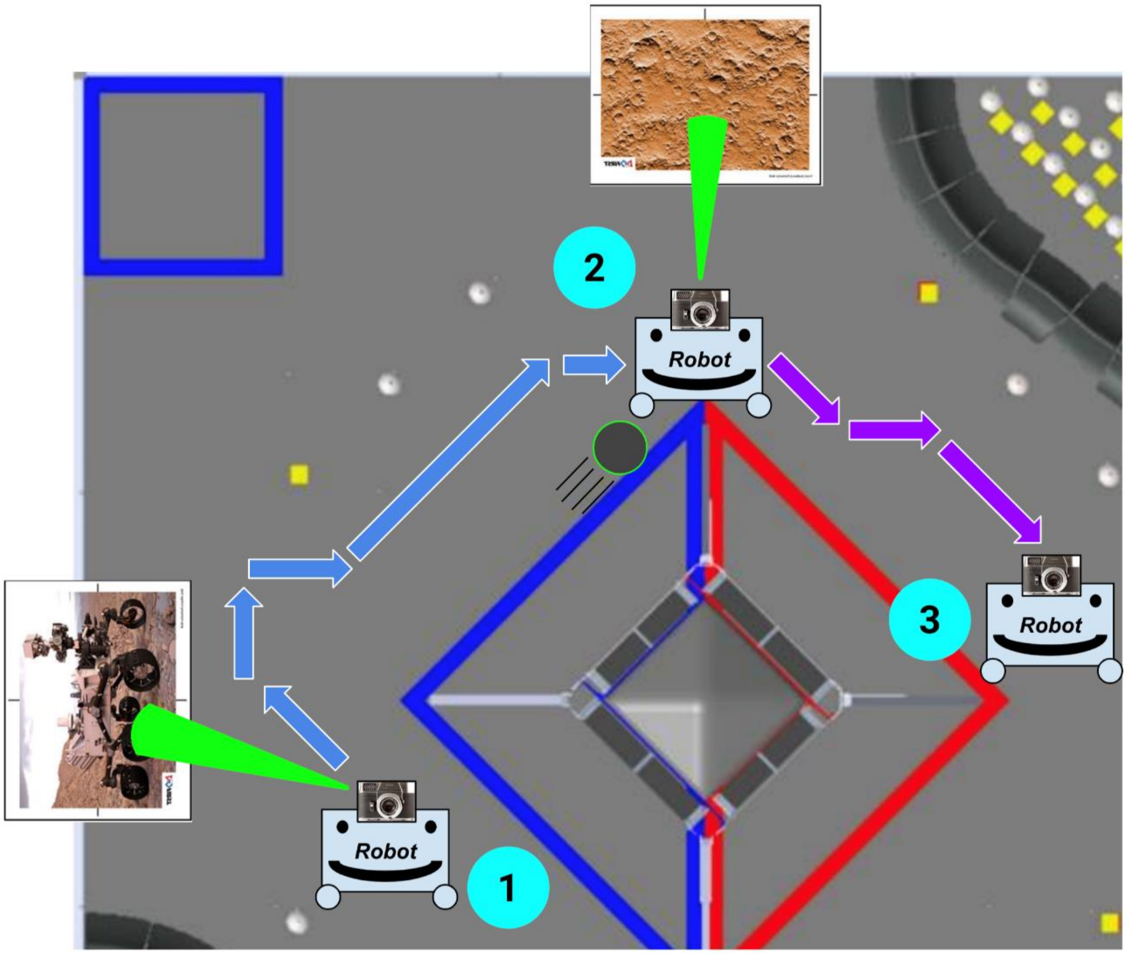
This system's simplicity becomes apparent when put into practice. For example, if an instruction says to move the robot North, the direction North's numerical value is 0. To rotate it by 90° clockwise, divide 90° by 45°, which equals 2. The 2 is added to North's numerical value, which results in 0+2 = 2. The 2 corresponds to the East direction, which is exactly a 90° clockwise rotation from North.

Realignment

Conceptualization and Implementation: C58, C59

As there is always a chance for error, such as another robot or debris in the way of a robot, a given robot might be knocked out of its planned path. This is another application for the Pathfinding Algorithms. During the course of following a path from the algorithm, the robot is always on the lookout for a new VuMark. (see *Vuforia Listener* in *Additional Summary Information*) If its camera sees one, it stops the current path it is on and restarts its pathing at the new VuMark. If the robot ever gets knocked off of its given path and fulfils the chance that it sees a new VuMark, the robot is able to get back on path.

The diagram below illustrates this process:



1. The robot localizes off of the Rover VuMark to figure out its (x, y) point
    a. Pathfinding Algorithms calculate a path to the destination (#3)
    b. Robot follows the pathing with encoders (blue arrows)
2. The robot is knocked off of its pathing by debris
    a. A new VuMark is seen and the robot stops its original pathing (blue arrows) and relocalizes, figuring out its new (x, y) point
    b. Pathfinding Algorithms calculate a path to the destination (#3)
    c. Robot follows the pathing with encoder (purple arrows)
3. Destination is reached

Error Correction & Fallback Plan B Routines

Throughout autonomous, there is opportunity for plenty of error to occur. Since a fundamental goal of autonomous is to have consistent, reproducible results, we try to better handle some errors that may result in a deviation from any planned autonomous route.

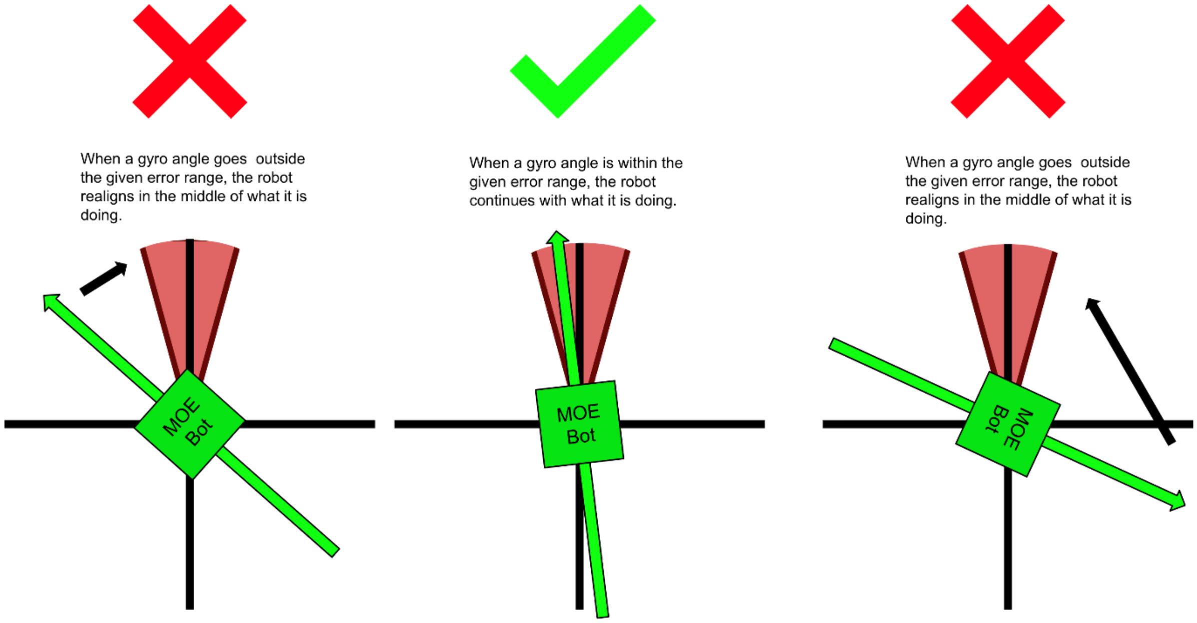
- Turn Corrections
- Distance Sensor Fallback

Turn Corrections

Implementation: C58, C59

In many of the routines and paths taken during autonomous, due to the nature of our mecanum wheels and the weight distribution of the robot, the robot gradually turns out of place. For example, when strafing normally for a period of 5 seconds, the robot could possibly turn 3° away from its starting angle. Over time, this error accumulates, and when sufficient, results in a faulty autonomous. To cut back on this, we define a given angle error range for any non-turning movement in autonomous. If the robot deviates from this error range, it interrupts what it is doing to turn back into proper position.

The diagram below shows the turn correction process.


When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.
When a gyro angle is within the given error range, the robot continues with what it is doing.
When a gyro angle goes outside the given error range, the robot realigns in the middle of what it is doing.
MOE Bot
MOE Bot
MOE Bot

Distance Sensor Fallback

Due the occasionally unreliable nature of the distance sensors, there is a need for a fallback when they produce errors. When getting readings in the middle of autonomous, a distance sensor sometimes gives wildly out of range readings or errors. To fix this, we add a fallback. If the sensor does not give reasonable values within 2 seconds, the robot uses a preprogrammed point to plug into the Pathfinding Algorithms rather than a more accurate point from the distance sensors.

Pure Pursuit Controller

With the creation of our pathfinding algorithms *(see sections above),* we were running into issues on how to properly follow the points the pathfinding suggested. Our original approach was to follow the points *directly*, meaning that the robot would directly travel one of eight directions. (North, East, Southeast, etc...).

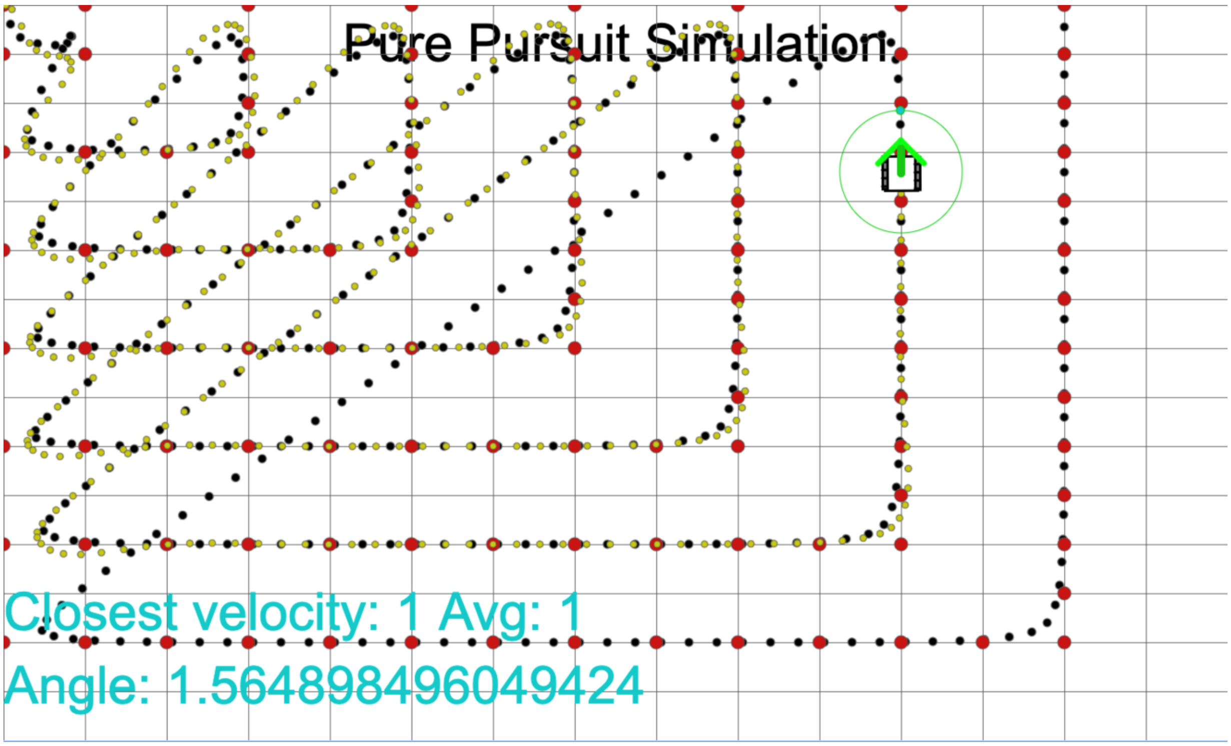
Instead, through the use of the Pure Pursuit Controller, we can smoothly follow points with a Tank Drive chassis. Although we have a Mecanum chassis, it can be adapted into a Tank Drive chassis when ignoring its strafing capabilities. Additionally, disregarding the use of strafing will allow the robot's movements to be more consistent, as strafing has more variable motion.

To be able to use the Pure Pursuit Controller, the following prerequisites must be met:

- Continual localization (or knowing exact x,y location on the field at any moment)
  - We accomplished this through odometry wheels
- Ability to accurately set velocities of wheels
  - We accomplished this through having encoders on all 4 drive wheels & using velocity *PIDs* for each one
- Path to follow
  - We accomplished this through two ways:
  - 1. Predefined route
  - 2. Jump Point Search pathfinding algorithm for dynamic pathing

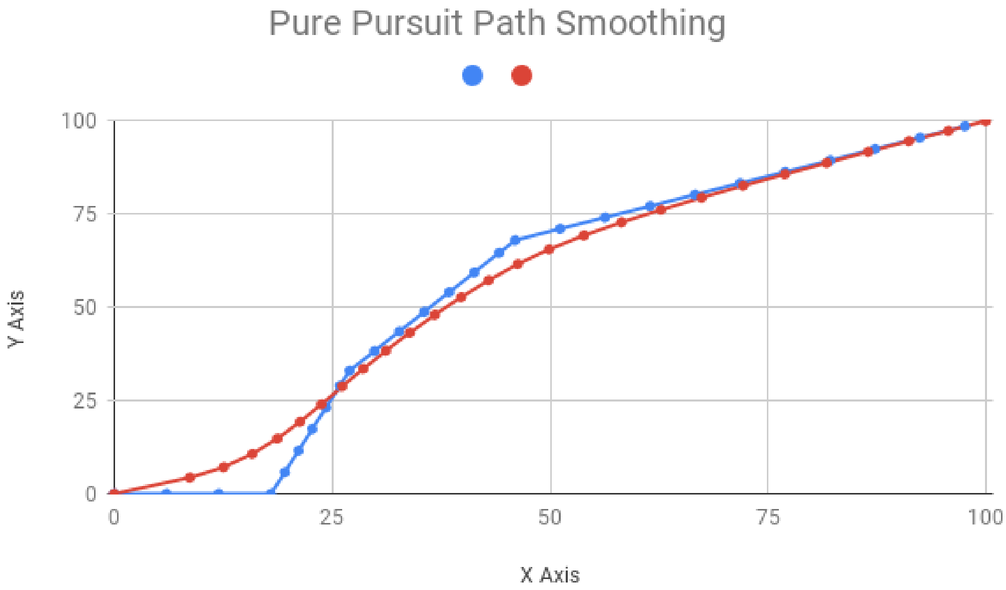
Through these prerequisites, Pure Pursuit is able to accurately correct itself whenever the robot deviates from the given path while still following the given path accurately.

Before doing Pure Pursuit in TeleOp, we also ran simulations to make sure that it worked. A picture of the simulation is seen below:

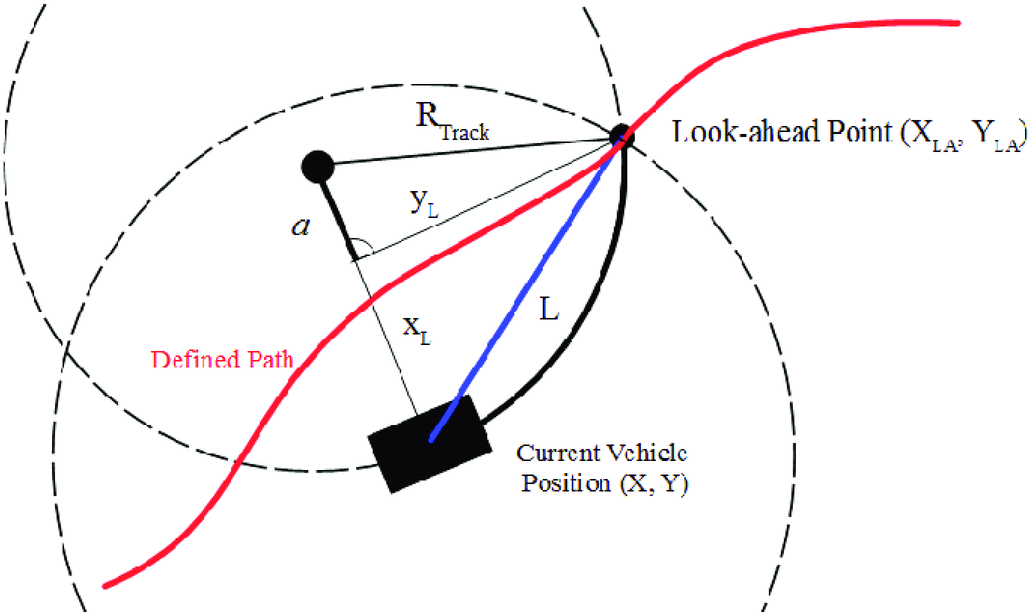


Path Following

The initial steps Pure Pursuit takes are as follows:



1. Take in a set of points – this is known as the pathing (blue line)
2. Inject in additional points between the given points (blue line)
3. Smooth all the points (red line)

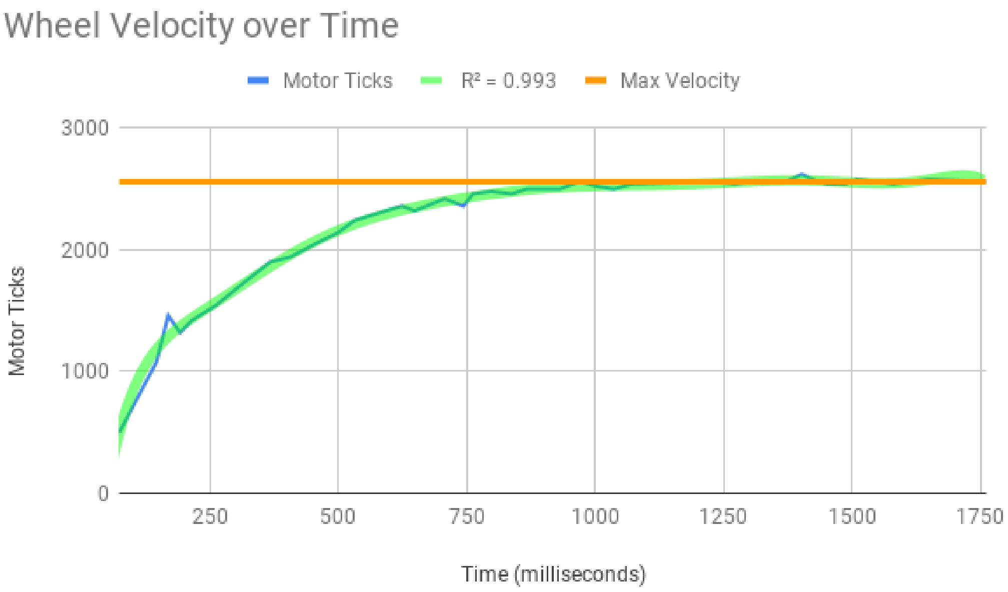


Pathfinder – Development of Automated Guided Vehicle for Hospital Logistics - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-Pure-Pursuit-algorithm-principle-Knowing-the-current-robot-location-and_fig6_319714221 [accessed 20 Apr, 2019]

After establishing a smoothed path to follow, the next general steps are:

1. Find the closest point on the pathing to the robot's current position
2. Use the next point on the pathing to establish a line with the closest point
3. Using the robot's current position and a *lookahead distance*, draw a circle around the robot and find its intersection with the line in #2 – this intersection will be known as the *lookahead point*
4. Using the *lookahead point*, calculate the robot's signed (+/-) curvature
    a. This signed curvature value lets the robot know how much to turn by
5. Using the curvature, calculate the velocity for the left side & right side of the robot
6. Apply the velocity to the wheels
7. **REPEAT STEPS 1-6** until the robot reaches the final destination

When calculating the velocity to apply to the wheels, Pure Pursuit needs to know the robot's maximum velocity. We measured this in ticks/time, which we then converted into our own custom units that we used for Pure Pursuit.

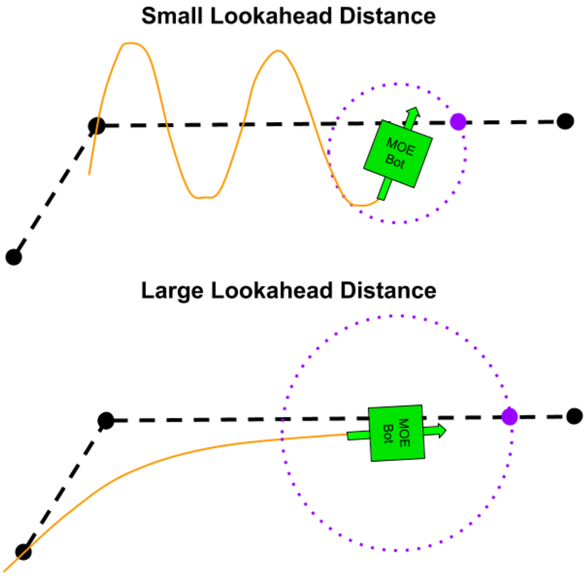


The orange line above shows the max velocity of the robot.

Constants

For Pure Pursuit to work properly, the following constants have to be tuned accordingly:

Lookahead Distance: How far the robot looks ahead from its current position for a point to follow. (and calculate curvature from)



As seen above, a small lookahead distance (small purple circle) results in short, choppy movements of the robot when following a path, while a large lookahead distance (larger purple circle) results in overgeneralized movements of the robot when following a path. A large lookahead distance might be dangerous in that it may hit a mineral since it cuts corners on turns too heavily. A proper lookahead distance will find a balance that allows the robot to accurately follow a path.

Max Velocity: The maximum velocity the robot can move at.

Turning Constant: How sharply the robot should turn

Smoothing A / Smoothing B / Smoothing Tolerance: These all control how closely the smoothing should follow the given path. In other words, whether the curves should closely fit the turns or generalize and approximate the turns instead.

Track Width: The horizontal width of the robot, that helps in the calculation of turns for the robot. Needs to be calibrated slightly higher than the actual width because turns are important for the Rover Ruckus game.

Overall, the Pure Pursuit Controller allows for fast, precise, and accurate autonomous path following.

Sampling Algorithm

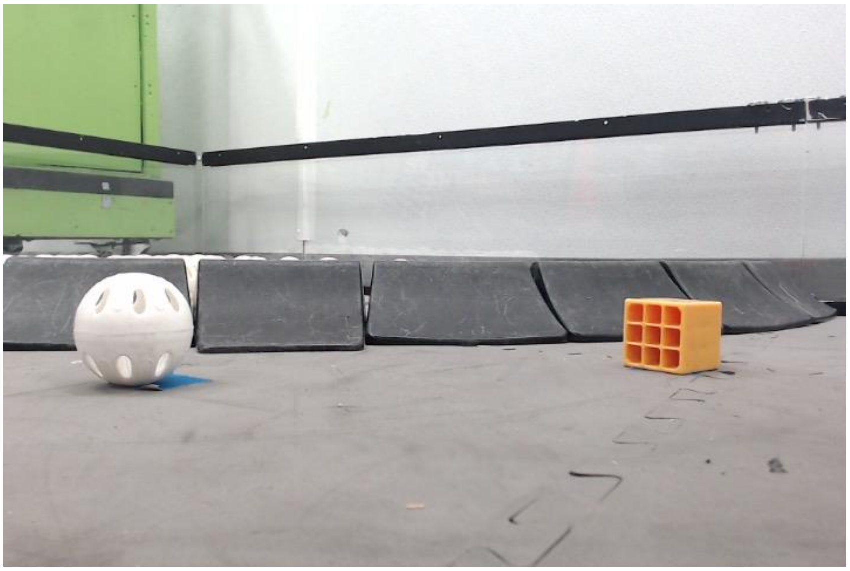
Our team tried using a variety of sampling methods, including the official TensorFlow neural network included in the SDK, creating our own neural network (see *Additional Summary Section*), and OpenCV for color detection. However, one key flaw with all of these methods was inconsistency. Even though the official TensorFlow network worked most of the time, we found that when placed on a competition field with a *yellow background* (like a gym floor), gold minerals were sometimes improperly recognized. This led to complications during competitions. Due to the flaws with the official TensorFlow network, our robot often picked the wrong mineral to sample.

To solve this issue, we wrote our own **custom algorithm** for the sake of sampling.

Steps:

1. Take camera data from Webcam

Example of camera data

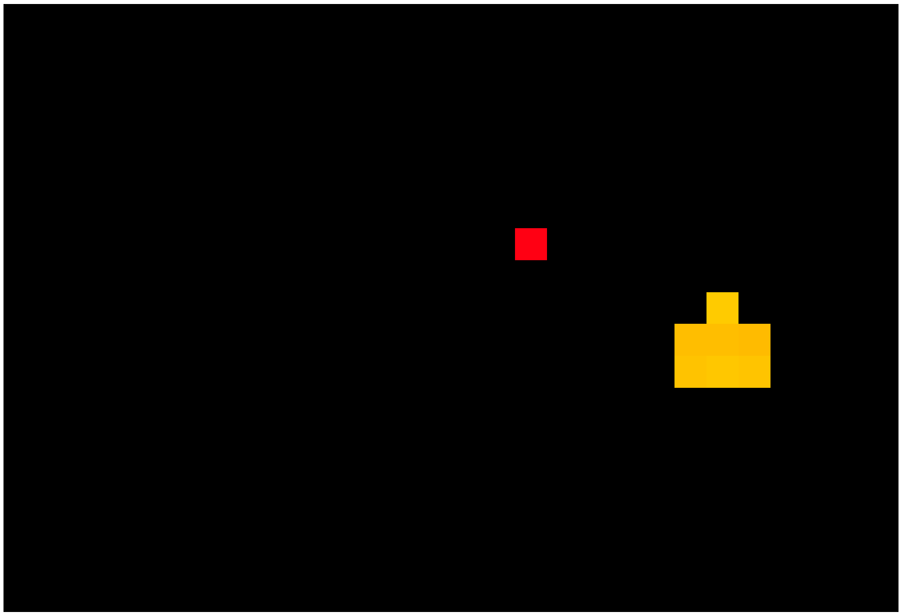


2. To remove any small splotches of yellow in the background (or any other strange color), reduce the resolution of the image.

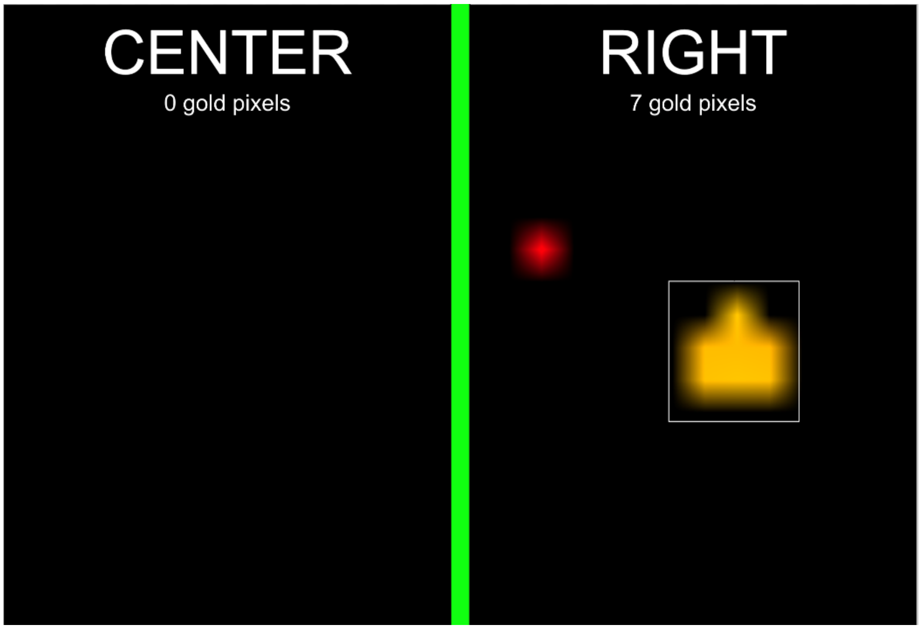
Reduced resolution of original image

3. Convert each pixel from RGB (Red, Green, Blue) color scheme to HSV (Hue, Saturation, Value) color scheme. Disregard any pixels without a high enough saturation value (< 0.5).

Visualization of image after pixels below HSV threshold are removed.



4. Compare amount of gold pixels in left and right side of image. The side that contains more gold pixels is considered the location of the gold sampling mineral. If both sides have less than 2 gold pixels, then the gold mineral is considered to be in the left location. (see *Gold Mineral Decision Algorithm* in *Additional Summary Section*)


CENTER
0 gold pixels
RIGHT
7 gold pixels

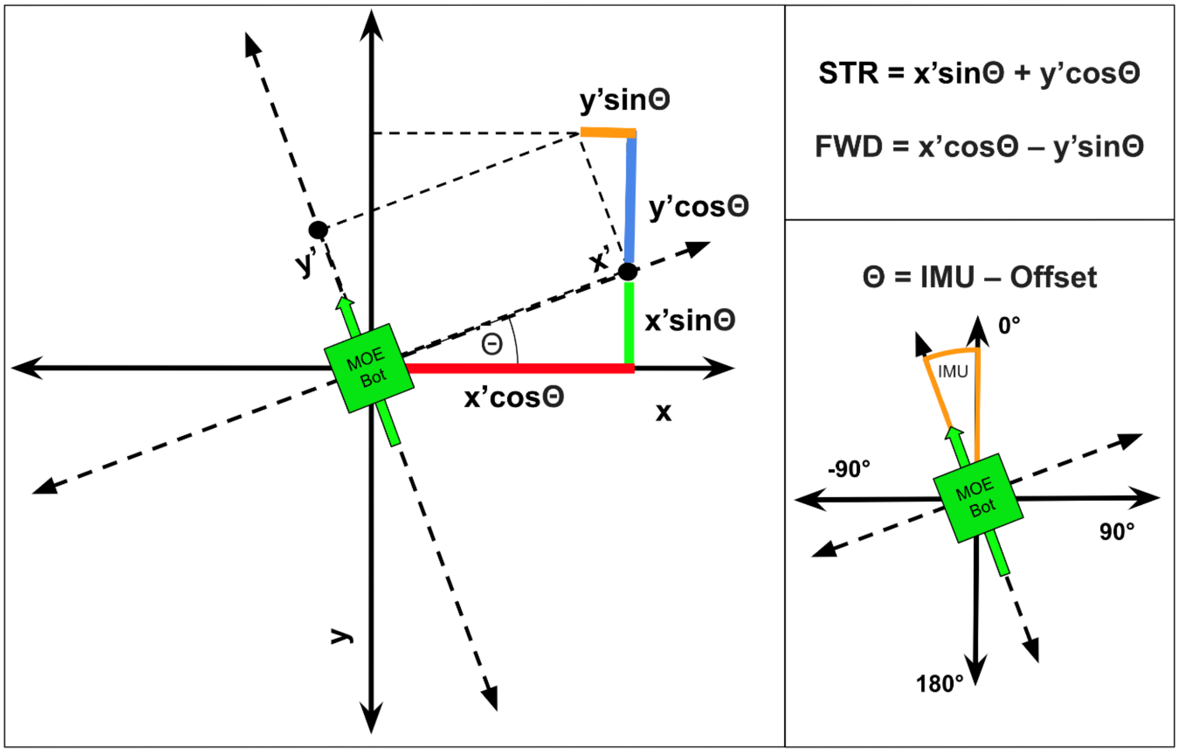
Driver Controlled Enhancements

Adjustable Field-Centric Movement

Conceptualization and implementation: C130, C131, C141

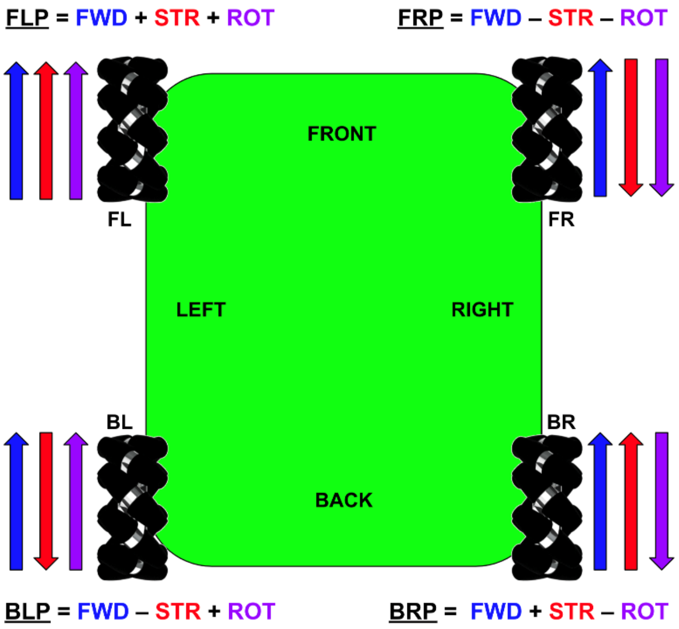
Since our robot uses a mecanum drive, it is capable of moving in any direction. Because of this, we are able to use **field-centric** motion rather than **robot-centric** motion. For the ease of the driver, we make all movements relative to the field rather than relative to the robot.

Note: In this diagram x' = Left Joystick X AND y' = Left Joystick Y



This diagram represents the **rotation of axes** that underlies the principles in field centric movement. The Θ is taken from the IMU, so that angle measurements are exact in the **field-centric** motion. Inputs for x' and y' are taken from the left joystick, and the right joystick x controls the rotation of the robot. Also, the offset for the IMU is to allow the driver to set a custom 0° point for the robot.

In the below diagram, the arrows indicate which direction the wheel turns when given a positive value for FWD (forward), STR (strafe), or ROT (rotation).



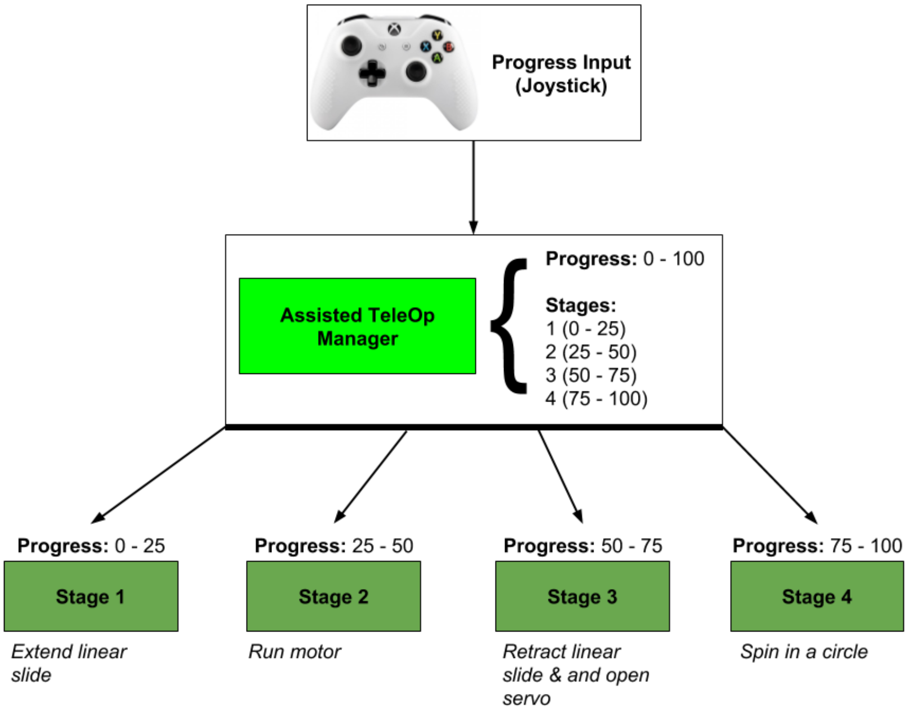
The corresponding values of FLP, FRP, BLP, and BRP are fed into the motors based on controller input so that field-centric movement occurs.

Assisted TeleOp

Our robot uses "**Assisted TeleOp,**" in which the robot performs a series of actions in one press of a button.

To make our Assisted TeleOp functions adaptable, we created an AssistedTeleOpManager class that takes in an AssistedConfig. This allows us to simply write a configuration for an automated task, immediately and quickly integrating it into our code.

Each of our Assisted TeleOp configs uses a progress variable that can be increased or decreased to control which stage of action it belongs to. This brings flexibility into the AssistedTeleOp, since the progress variable could be automatically increased ***or*** be controlled by a single joystick. This heavily simplifies complex motions that would otherwise require many controller inputs.



Looking at the diagram above, the manager abstracts much of the logic of progressing through an Assisted TeleOp routine. By using the manager, moving forwards and backwards with an assisted routine becomes very simplified. All the programmer that uses the manager has to do is set the *progress* variable that controls the motion of the robot.

This infrastructure also rapidly speeds up development of Assisted TeleOp routines, since the programmer only has to focus on the logic of the Assisted TeleOp, not the mundane transitions between stages of Assisted TeleOp.

For example, our transfer mechanism, which sends the minerals from the harvester to the dispenser (and involves complex movements), can use this infrastructure to manage progress through stages.

Lander Based Movement

Conceptualization and implementation: C141

Our robot also has “**lander specialized”** movement that allows for fine turning before hanging the robot. The D-pad allows for smaller adjustments in four-directional movement corresponding to the four buttons on the D-pad. Also, the bumpers allow for smaller adjustments in turning to fix orientation before raising the lift to hang.

Controls

Gamepad 1

Left Joystick:

Left Joystick *Down*: Toggle between field-centric and robot-centric movement

Right Joystick *Left, Right*: Turn robot

A: Toggle dispenser orientation towards crater or within robot

X: Toggle dispenser orientation up or down

B: Toggle end game mode

Y: Reset 0° point (forward heading) for field-centric movement

Left & Right Triggers: Raise and lower hanging lift

Left & Right Bumpers: Fine turning (see *Lander Based Movement*)

D-Pad: (see *Lander Based Movement*)

> **UP** → Move slowly forward
>
> **DOWN** → Move slowly backward
>
> **LEFT** → Strafe slowly left
>
> **RIGHT** → Strafe slowly right

Gamepad 2

Left Joystick *Click*: Open and close intake gate & lower harvester servo

Left Joystick: Extend and retract harvester linear slide

Left & Right Triggers: Controls green intake wheel that brings in minerals

D-Pad: (see *Lander Based Movement*)

> **UP** → Move slowly forward
>
> **RIGHT** → Strafe slowly right

Autonomous Routines

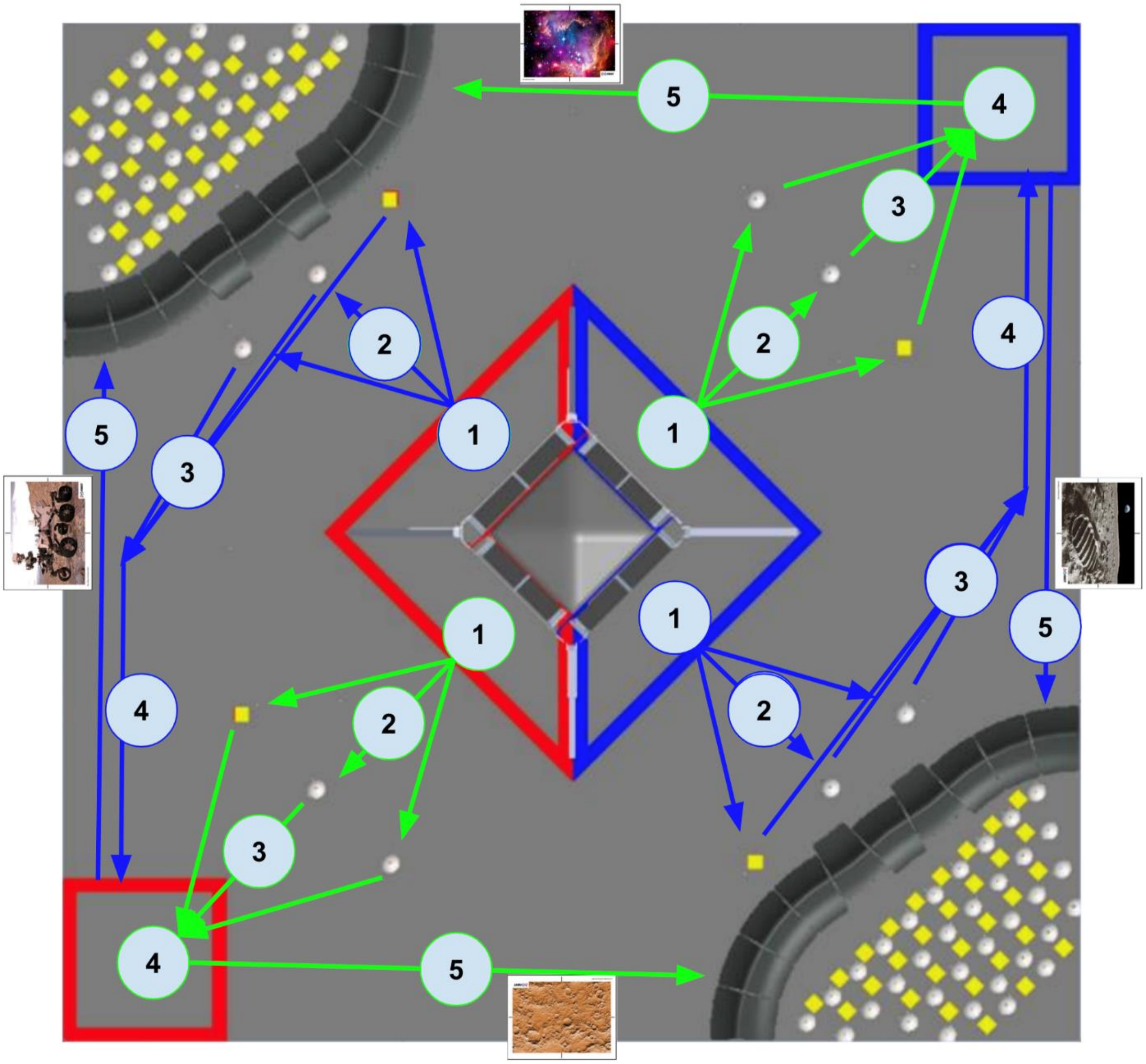
Conceptualization and testing of routine: C76

Iterative improvements, testing, and debugging: C84, C85

Testing environment fabrication: C104

2nd Stage Iterative improvements, testing, and debugging: C105, C106, C121, C122

Testing and evaluations: C140, C141, C145



The image above represents our autonomous routine for all possible starting points. Regardless of starting point, our autonomous strives to accomplish delatching, sampling, depositing team marker, and parking. We plan on executing the following steps for each autonomous:

1. Landing & Detecting gold mineral
2. Travelling to and knocking off gold mineral
3. Travelling to depot
4. Deposit the Team Marker and leave gold mineral
5. Travel to crater & extend arm into crater

Initialization

1. Initialize motors, servos, sensors, and all other devices from the Hardware Map.
2. Initialize REV IMU sensor
3. Initialize Vuforia
4. Reset Crater Extension Arm to position 1
5. Have robot say “Finished initialization” through speakers to doubly confirm initialization

Steps

1. Landing & Detecting gold mineral (30 pts.)

1. Use linear actuator to move lift up for given # of encoder tics
    a. The robot touches the floor and the claw goes above the top of the handle on the lander
2. Using multithreading, lower the lift back to its starting position
    a. (see *Multithreading* in *Key Algorithms*)
    b. The program continues on without waiting for the process to finish because of the multithreading
3. Turn ~70° to see 2 minerals and decide which is gold
    a. (see *Sampling Algorithm* in *Key Algorithms*)
4. The robot turns the appropriate # of degrees to face the gold mineral in left, right, or center

2. Landing & Detecting gold mineral (25 pts.)

1. Move forward appropriate # of inches to knock off gold mineral from its starting position
2. Continue moving forward to safely clear other 2 silver minerals

3a. Travelling to depot (Depot Start Point)

1. Turn appropriate number of degrees to face VuMark (Rover/Moon)
2. Move forward appropriate # of inches to read data from the VuMark (distance, angle, etc...)

3b. Travelling to depot (Crater Start Point)

1. Turn appropriate number of degrees to face the depot
2. Move forward appropriate # of inches to reach the depot
3. Localize and figure out (x, y) position on the MOEPS global field grid by using Vuforia
4. Calculate and follow path to depot using Pathfinding Algorithms

4. Deposit the Team Marker (15 pts.) – If Depot Start Point, deposit gold mineral in depot (+2 pts.)

1. Turn appropriate # of degrees for front of robot to face the front/back wall
2. Localize and figure out (x, y) position on the MOEPS global field grid by using distance sensors
    a. (See *Distance Sensor Localization* in *Localization*)
    b. If the distance sensors have an error in measurement, fallback to Plan B
3. Turn appropriate number of degrees for left side of robot to face corner of field
4. Drop off Team Marker
5. Calculate path to crater using Pathfinding Algorithms
    a. ***Plan A:*** Use (x, y) position from distance sensors
    b. ***Plan B:*** Use default (x, y) position as an estimate – less accurate than Plan A
    c. (see *Jump Point Search/A*/Dijkstra's Pathfinding Algorithms*)

5a. Travel to crater & extend arm into crater (10 pts. – Depot Start Point)

1. Turn ~90° right for the back to face the crater
2. Apply 90° rotation to pathfinding algorithm results to fit new robot orientation
    a. (see "*Rotational Symmetry" in Jump Point Search/A*/Dijkstra's Pathfinding Algorithms*)
3. Follow pathing to reach crater on other alliance's side
4. Extend arm
5. Drive backwards to guarantee arm is in crater

5b. Travel to crater & extend arm into crater (10 pts. – Crater Start Point)

6. Follow pathing to reach crater
7. Extend arm
8. Drive backwards to guarantee arm is in crater

Additional Summary Information

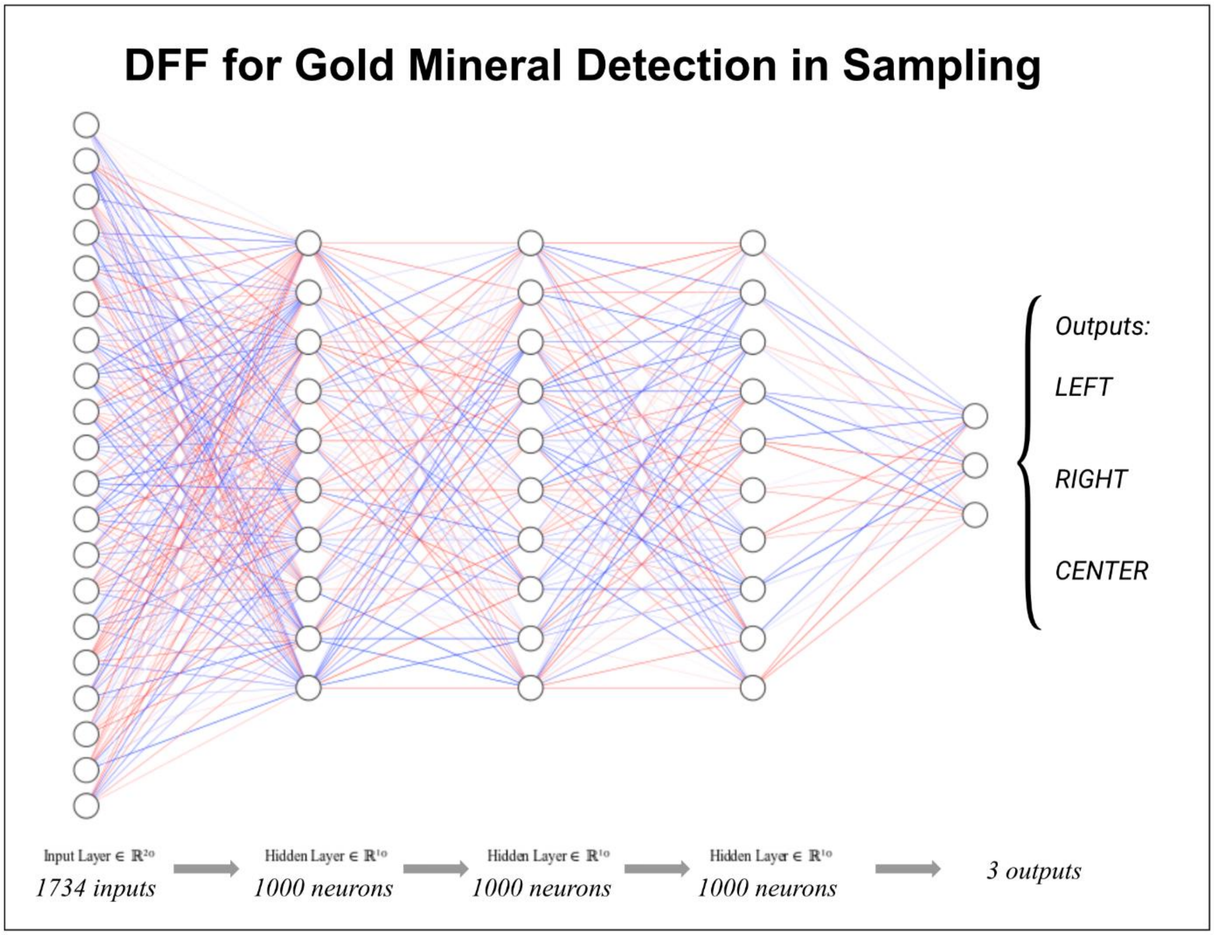
Creating An Artificial Neural Network (ANN)

Conceptualization and implementation: C110, C111, C112, C113, C114, C115

Before the integration of TensorFlow Lite into the official FTC App, we created a neural network with TensorFlow that could distinguish the left, center, or right position of the gold in the sampling minerals.

Choosing the Correct Structure

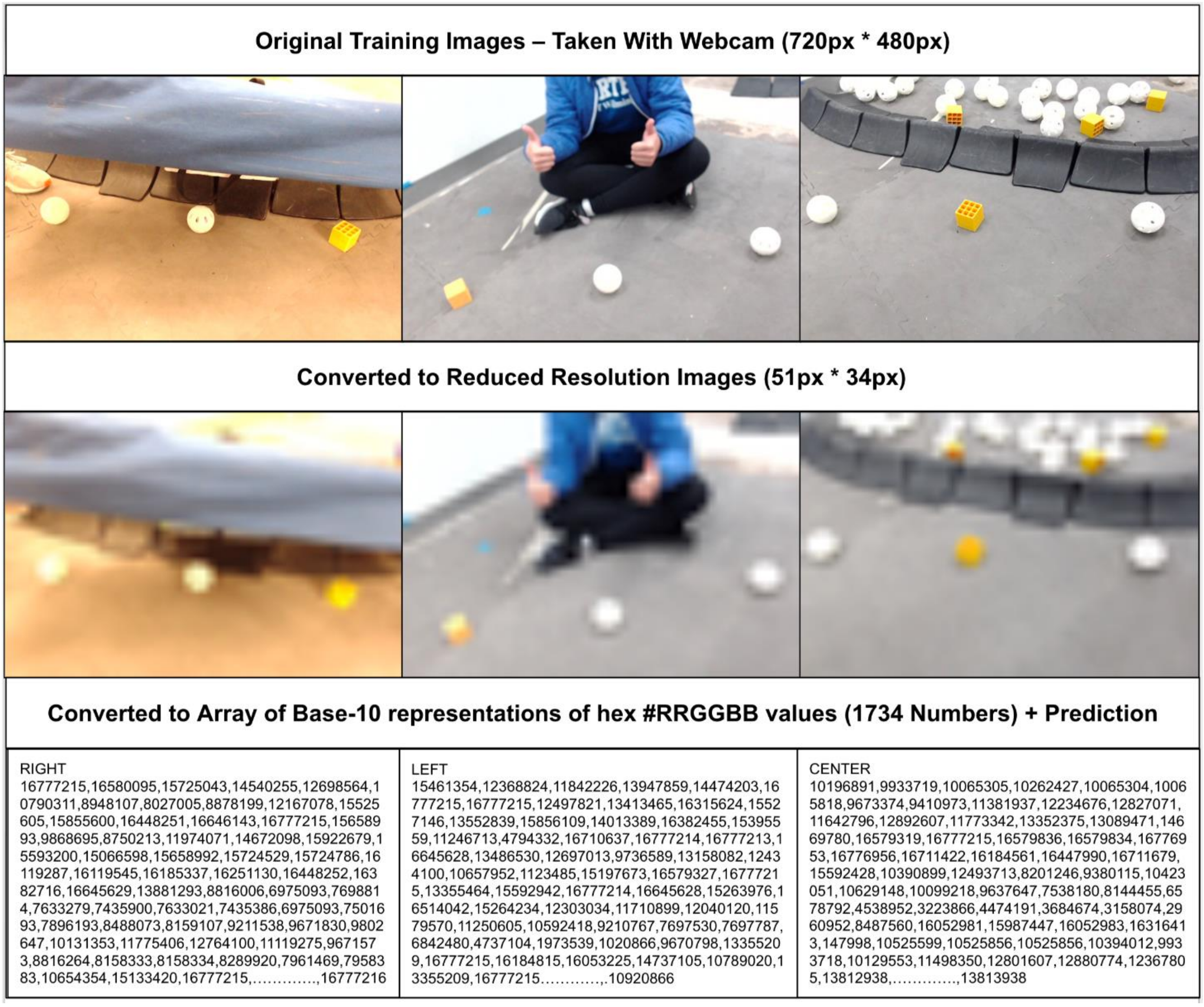
We decided to go with a ***deep feed-forward (DFF) neural network*** with ***backpropagation,*** a commonly used technique to train a neural network based around gradient descent.



There are 1734 inputs from the 1734 pixels in each of our input images, and 1000 neurons in each hidden layer. The final output has 3 possibilities. This neural network requires less training data because the problem at hand is fundamentally clear in terms of processing; there are no

complex edge detections required. All the neural network has to do is distinguish between yellow and white and their locations in the images.

Acquiring Training Data & Preprocessing (Total of 48 Images)



Original Training Images – Taken With Webcam (720px * 480px)

Converted to Reduced Resolution Images (51px * 34px)

Converted to Array of Base-10 representations of hex #RRGGBB values (1734 Numbers) + Prediction

RIGHT	LEFT	CENTER
16777215,16580095,15725043,14540255,12698564,10790311,8948107,8027005,8878199,12167078,15525605,15855600,16448251,16646143,16777215,15658993,9868695,8750213,11974071,14672098,15922679,15593200,15066598,15658992,15724529,15724786,16119287,16119545,16185337,16251130,16448252,16382716,16645629,13881293,8816006,6975093,7698814,7633279,7435900,7633021,7435386,6975093,7501693,7896193,8488073,8159107,9211538,9671830,9802647,10131353,11775406,12764100,11119275,9671573,8816264,8158333,8158334,8289920,7961469,7958383,10654354,15133420,16777215,.............,16777216	15461354,12368824,11842226,13947859,14474203,16777215,16777215,12497821,13413465,16315624,15527146,13552839,15856109,14013389,16382455,15395559,11246713,4794332,16710637,16777214,16777213,16645628,13486530,12697013,9736589,13158082,12434100,10657952,1123485,15197673,16579327,16777215,13355464,15592942,16777214,16645628,15263976,16514042,15264234,12303034,11710899,12040120,11579570,11250605,10592418,9210767,7697530,7697787,6842480,4737104,1973539,1020866,9670798,13355209,16777215,16184815,16053225,14737105,10789020,13355209,16777215............,.10920866	10196891,9933719,10065305,10262427,10065304,10065818,9673374,9410973,11381937,12234676,12827071,11642796,12892607,11773342,13352375,13089471,14669780,16579319,16777215,16579836,16579834,16776953,16776956,16711422,16184561,16447990,16711679,15592428,10390899,12493713,8201246,9380115,10423051,10629148,10099218,9637647,7538180,8144455,6578792,4538952,3223866,4474191,3684674,3158074,2960952,8487560,16052981,15987447,16052983,16316413,147998,10525599,10525856,10525856,10394012,9933718,10129553,11498350,12801607,12880774,12367805,13812938,.............,13813938

Key ideas in images for the neural network:

- Changed background of images to show background does not matter
- Changed lighting of images to show lighting does not matter
- Changed tilt of images to show tilt does not matter
- Did not change order of minerals to show **only** ordering of minerals matter

NOTE: The preprocessing was automated using Python scripts to save time.

The reason the images' resolutions were reduced was due to the fact that training the network would require more time. Although training the network at full resolution would be fine, it

would possibly take a few minutes, and this can get cumbersome when refining and tweaking the data. We felt predictions could be made just as well at reduced resolution.

The process of converting to a Base-10 representation of hex #RRGGBB values:

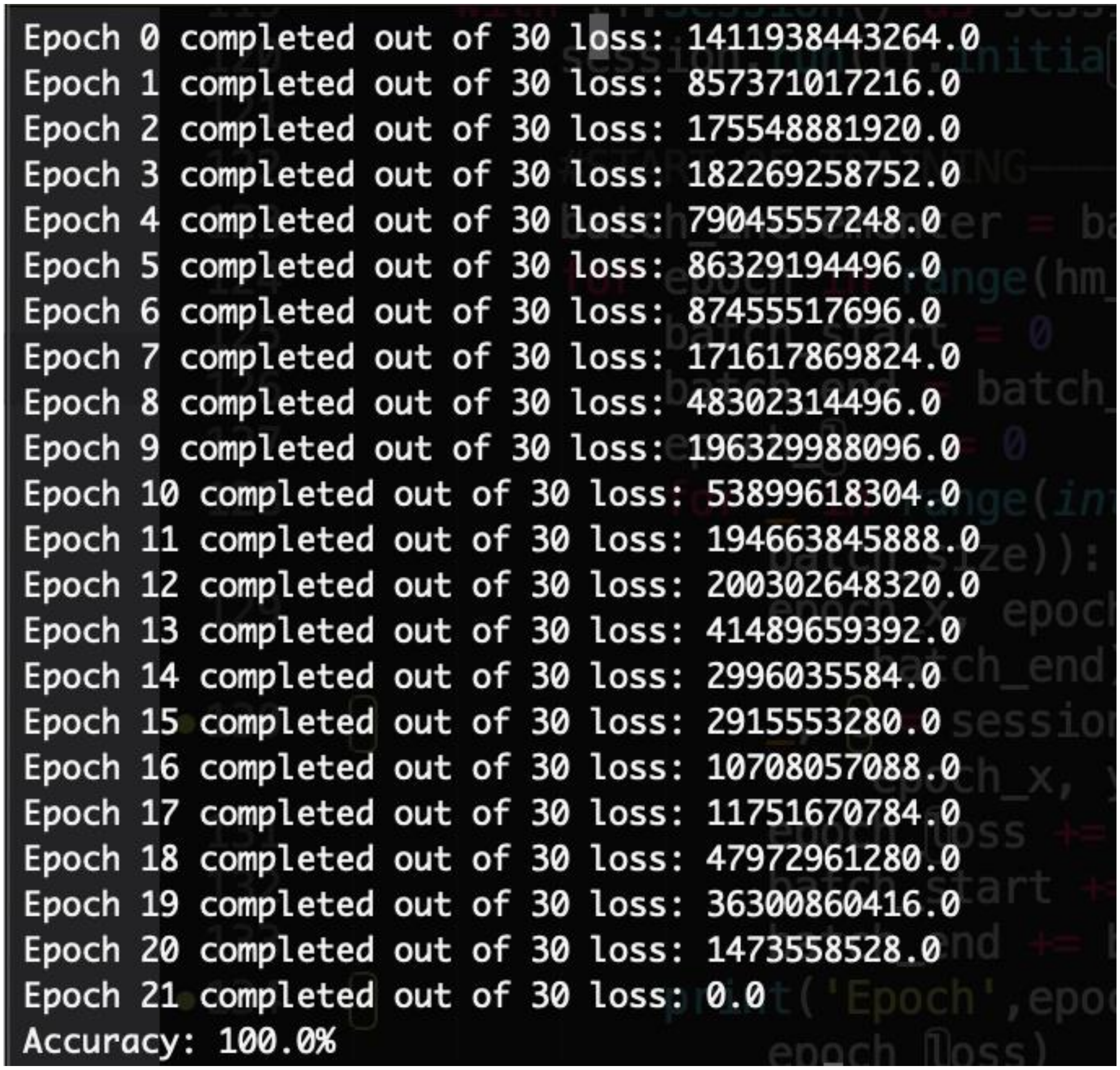
1. Take individual R, G, B (0 – 255) values of each pixel
2. Convert R, G, B values into one hexadecimal (base-16) number (#RRGGBB)
3. Take hexadecimal number #RRGGBB into a decimal (base-10) number

All the data was saved to a .txt file to be trained on later.

Training & Accuracy of Neural Network

Because we reduced the resolutions of the images in the preprocessing, training time for the 48 images was incredibly short: 5-15 seconds.

After training a successful model, here were our results.



```
Epoch 0 completed out of 30 loss: 1411938443264.0
Epoch 1 completed out of 30 loss: 857371017216.0
Epoch 2 completed out of 30 loss: 175548881920.0
Epoch 3 completed out of 30 loss: 182269258752.0
Epoch 4 completed out of 30 loss: 79045557248.0
Epoch 5 completed out of 30 loss: 86329194496.0
Epoch 6 completed out of 30 loss: 87455517696.0
Epoch 7 completed out of 30 loss: 171617869824.0
Epoch 8 completed out of 30 loss: 48302314496.0
Epoch 9 completed out of 30 loss: 196329988096.0
Epoch 10 completed out of 30 loss: 53899618304.0
Epoch 11 completed out of 30 loss: 194663845888.0
Epoch 12 completed out of 30 loss: 200302648320.0
Epoch 13 completed out of 30 loss: 41489659392.0
Epoch 14 completed out of 30 loss: 2996035584.0
Epoch 15 completed out of 30 loss: 2915553280.0
Epoch 16 completed out of 30 loss: 10708057088.0
Epoch 17 completed out of 30 loss: 11751670784.0
Epoch 18 completed out of 30 loss: 47972961280.0
Epoch 19 completed out of 30 loss: 36300860416.0
Epoch 20 completed out of 30 loss: 1473558528.0
Epoch 21 completed out of 30 loss: 0.0
Accuracy: 100.0%
```

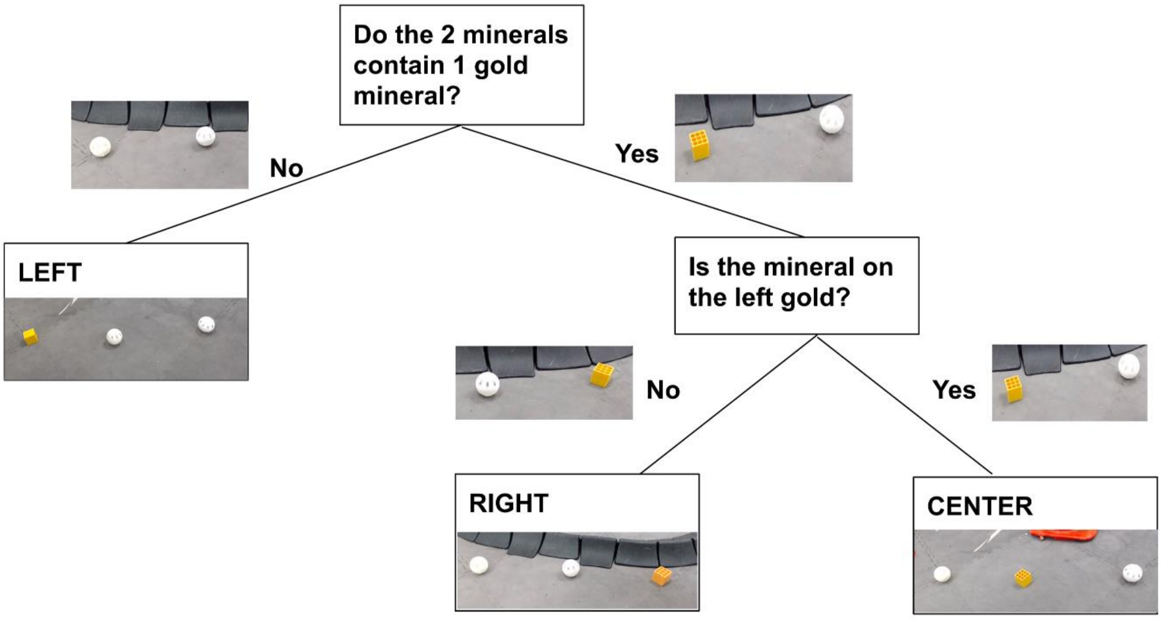
As shown on the image, the accuracy is 100% on our 48 images. The loss of 0.0 might be an indication of over-fitting the training data, but the network was able to successfully predict our test data, so the network indicates it has not overfitted to the point of inaccuracy.

However, when put into practice, this neural network had decent accuracy (~60%), but definitely not do the point needed for reliability in autonomous..

Gold Mineral Decision Algorithm

See engineering notebook entries: C121, C122, C128, C129, C130

Since our robot was only able to see two minerals on the field, we had to write an algorithm to figure out which one is gold. To accomplish this, we guaranteed that the two minerals we saw would be the 2 right minerals out of the 3 in sampling. If the 2 were silver, the gold would be on the left. If the gold was the left of the 2 right minerals, it would be on the center. If the gold was on the right of the 2 right minerals, it would be on the right.


Do the 2 minerals contain 1 gold mineral?
No
Yes
LEFT
Is the mineral on the left gold?
No
Yes
RIGHT
CENTER

Vuforia Listener

(For the purpose of this, see *Realignment* in *A* Pathfinding Algorithm & Dijkstra's Algorithm*.)

To be able to retrieve the exact moment a Vuforia tag is seen by the camera, we took a unique approach in capturing these events. In Vuforia, there is a class known as the **VuforiaTrackableDefaultListener**. Normally, this class is called to check and retrieve a VuMark when one knows exactly when they will see a VuMark. However, due to the variable nature of our implementation of the A* and Dijkstra Pathfinding Algorithms (see *A* Pathfinding Algorithm & Dijkstra's Algorithm*), there is a need to know when a VuMark is found in a safe and efficient way.

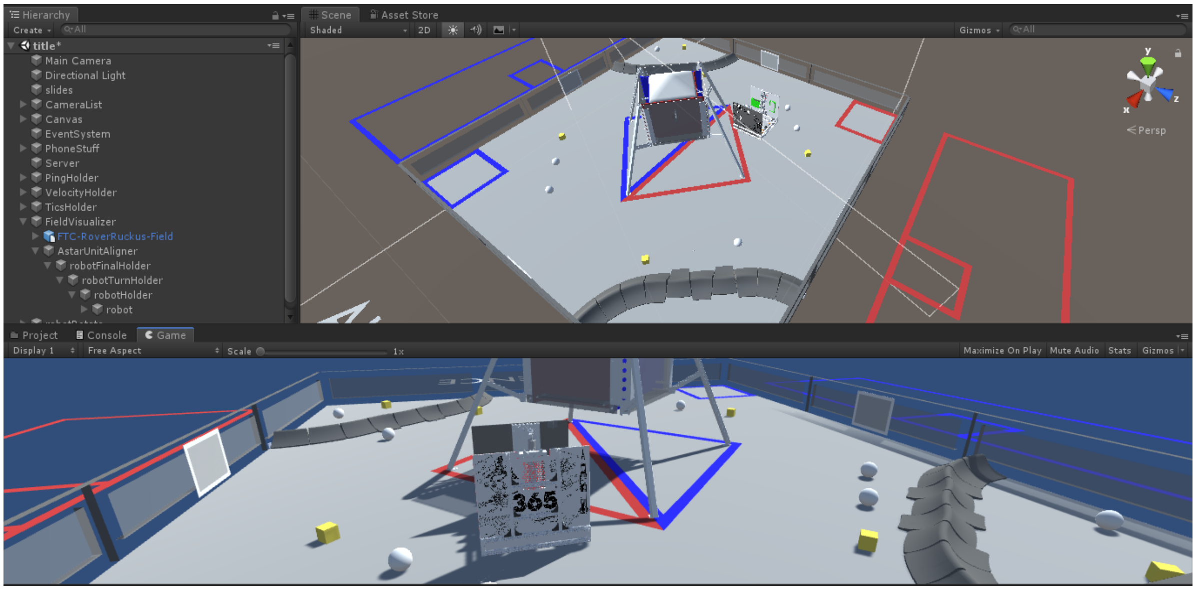
This approach creates a class called **MOEListener** that extends the **VuforiaTrackableDefault Listener,** of which is used instead of the **VuforiaTrackableDefaultListener**. When overriding the methods in the default listener, we realized that there was no convenient method to realize when a *new* VuMark was found, so we modified one of the existing methods to let us know when a new VuMark was found. This way, in the event of a new VuMark, our robot would be notified properly.

Since Vuforia runs on a separate thread, there needed to be way to guarantee that the program would work consistently on multiple threads while being able to share information between the two. For this purpose, we utilized **Atomic** variables in Java for thread-safe sharing of information between the threads. If we did not, there is the slight chance that when two threads modify the same variable at the same time, there could be a loss of information. The usage of **Atomic** variables notifies the robot in the middle of following the path found by the Pathfinding Algorithms to stop what it is doing and realign against the VuMark.

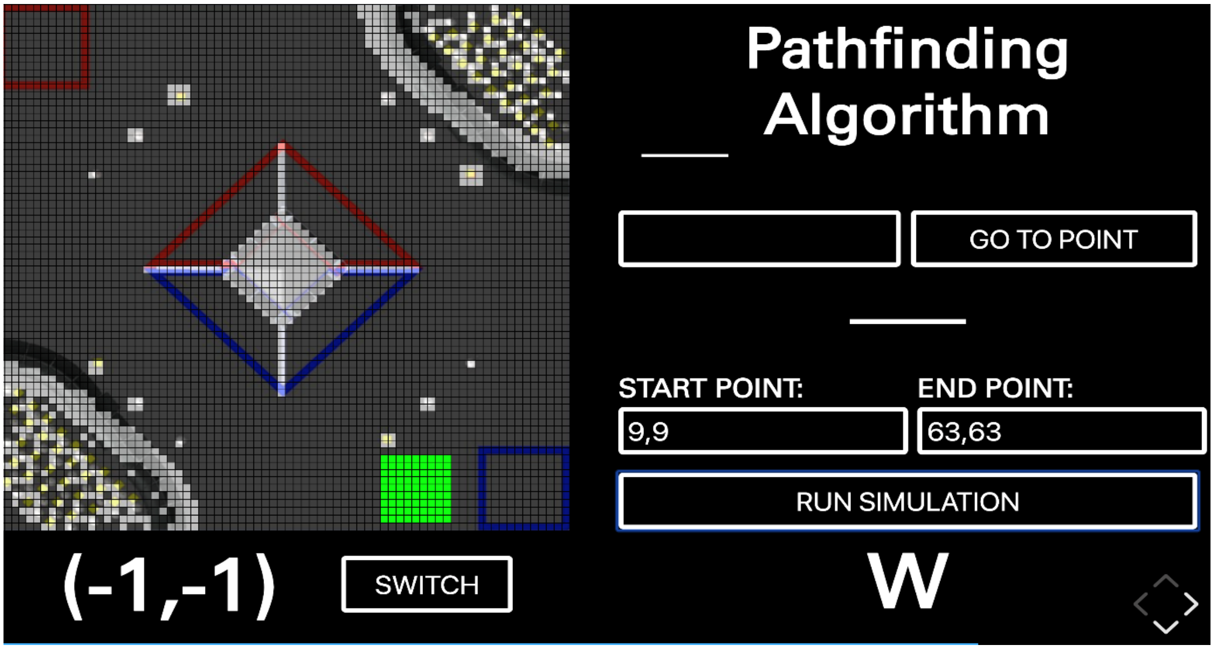
Simulations

Throughout the creation and testing of our code, we have used simulations to quickly and test the efficacy of our algorithms before putting them on the actual robot.

We have run simulations in a 3D environment with the *Unity* engine.



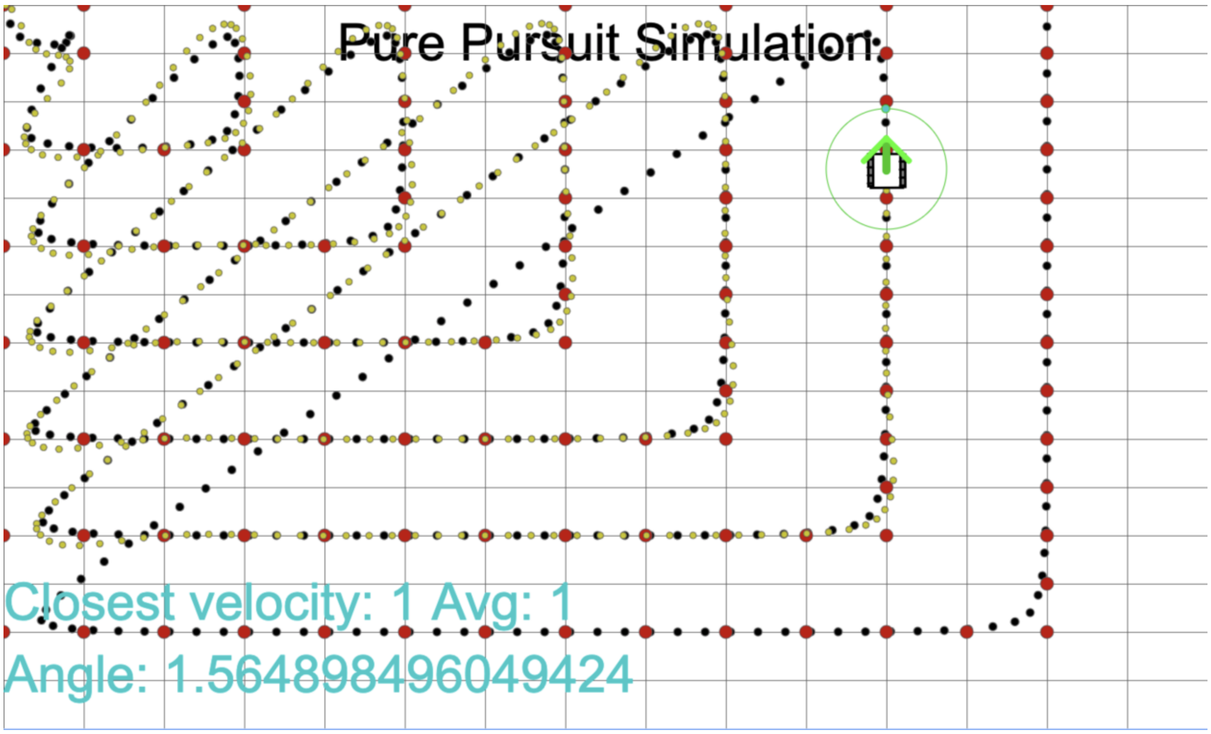
Additionally, we have run simulations for the testing of our *Pure Pursuit Controller* and *Pathfinding Algorithms*.



START POINT: 9,9

END POINT: 63,63

(-1,-1)



Pure Pursuit Simulation

Closest velocity: 1 Avg: 1

Angle: 1.564898496049424

Text-To-Speech (TTS)

For additional fun and utility, we incorporated the *Google Text-to-Speech* technology that allows text to be read aloud in a human-like fashion.

On the field, our robot likes to let us know how it is doing through a variety of phrases, including when it is initialized. Certain phrases include:

- "Initialized Vuforia"
- "Initialization Complete"
- "Initialized Gyro"

Over time, hearing these phrases can become cumbersome; however, our robot likes to spice things up. When completing certain tasks in autonomous, the robot likes to show its patriotism and loyalty to our team. Certain phrases it uses include:

- "Go MOE"
- "Whooo!"
- "Hi _____" (where _____ may be someone's name)
    - Note: this is pre-programmed, we have not yet integrated the facial recognition technology for the robot to detect people on its own

Our robot also has a fondness for music, which it may play on or off the field. More than anything else, our robot's personality makes interacting with it more interesting and fun! :)