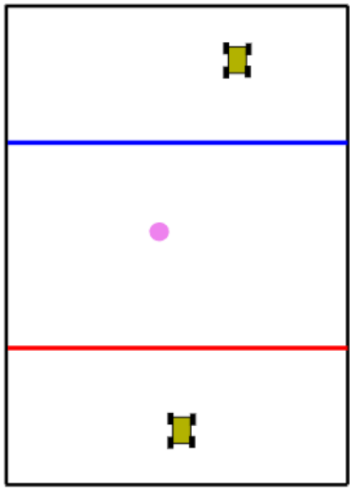
Final Project Report

Objective

The goal of the project is to build a robot that can play a simplified version of soccer using an IR ball and IR seeker sensors. Two teams will play against each other on a field that has 3 zones: two defense zones that only the defensive team's robot can be in and a middle zone that both teams' robot can be in. The goal for each team is to have the ball cross the other team's base line to score a goal. Once a goal is scored, the robots are moved into their team's defense zone and the ball is placed in the center. Then the game is started again with the team that has been scored on getting a 1s head start.



Files

- *main.py*: main code that allows the robot to ball search, dribble, and charge the ball.

Robot Design

The design for project 2's robot was the base for this project. The EV3 brick is still placed on a platform that is easily detachable, allowing for easy modification of the robot's, "internal" components. The constraints for this project required our robot to be compressed. The final dimensions of the robot were 9in* 8.5in * 6in. The soccer project is going to be highly physical; not only will walls be hit but various robots will also inadvertently hit ours during the match. To combat this, we added various stabilizing rods that connect from the outer edges of the robot to the main frame, making it more stable and sturdier. Additionally, we added redundant connections and frames all under the vehicle and on the platform as a backup in case of damage. These design choices have made the robot robust and impact resistant, while maintaining the agility needed for the competition. Making turns is an important part of this project so the wheels need to have enough grip on the ground to turn effectively, meaning a specific weight distribution is required. We moved the block further back on the platform to counteract the added weights of the sensors. The bumper was heavily modified in this project. We added pincer-like

protrusions from the bumper allowing us to hook onto and move with the ball. The gap between the pincers is wide enough to where we can successfully “kick” the ball, but also charge with it, while not letting it go.

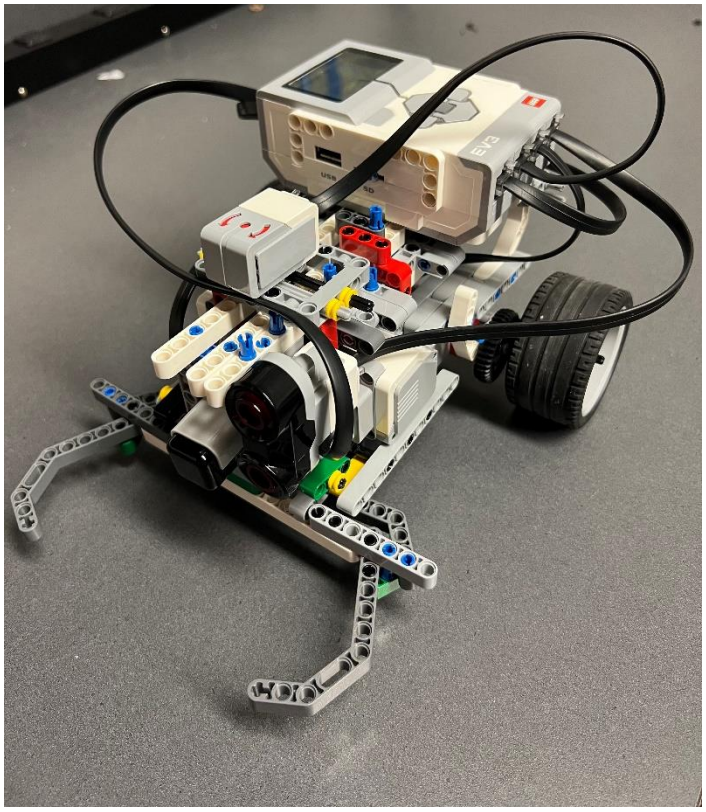
A HiTechnic IRSeeker Sensor is attached directly in front of the robot, over the bumper. We had to make sure that the sensor didn’t protrude too far over the bumper that it would prevent the IR soccer ball from being caught inside the bumper. The EV3 MicroPython code to read data off the sensor was provided by the TA.

An EV3 Ultrasonic Sensor is attached directly next to the IR sensor over the bumper. While the Ultrasonic sensor might interfere with the IR readings from the attached side, this was the best position the team found to get accurate ultrasonic readings directly in front of the robot. There was no room to place the ultrasonic sensor lower on the robot and if placed too high, it’s possible that the sensor would read false distances over the walls of the playing field.

The EV3 Color Sensor is attached behind the bumper facing towards the ground. It rests about 1cm above the surface of the floor. The sensor is used to signal to the robot that it crossed the line that marks the enemy team’s defensive zone.

The EV3 Gyroscopic sensor was added to the robot for the robot to know which side of the field it’s facing. We assume that the robot is always going to start facing straight towards the enemy goal and this angle is designed as 0 degrees. Since it’s possible for the gyroscope to have a reading greater than 360 (negatives included), the readings are always used after a “% 360” operation is used.

Group 21 - Michael Nguyen, Muhammad Muawiz Farooqi, Alfredo Reina Corona
Final Project Report



Ball Search Strategy

This function is a modified version of the “wander” functionality of project 2. The ball_search() function uses readings from the color sensor, Infrared sensor, and ultrasonic sensor to search for the ball. If the ultrasonic sensor detects the wall within ~15cm of the robot, it backs up and turns around, picking a random direction to turn in. If it does not detect the ball or a wall less than 15cm away, it runs the motors at a constant speed and then picks a random direction to turn in. After three random turns with no success finding the ball, it does a sweep action where it scans 20 degrees to right and left of its current orientation. This process will continue until the IR sensor picks up a reading from the ball or the color sensor detects the opponent's line. If the ball is detected, it will break out of the loop and enter the aggressive() function; if the opponent’s line is detected, it will back up, turn it will break out of the loop and enter the l.

Aggressive Strategy

This function includes all the behavior the robot is expected to perform while the IR seeker sensor detects the ball. Throughout the aggressive function, the IR-seeker sensor constantly updates its readings on whether it detects the ball. If at any point the sensor loses the ball’s signal, then the robot exits from the aggressive function and moves onto the ball_search strategy/function.

The aggressive function basically consists of multiple if-else statements to describe what behavior the robot should perform given a certain scenario. If the robot detects the IR-ball, while it's not directly in front, and at a distance where the received signal strength is less than 15, the robot will slowly orient itself until it's directly facing the ball. If the robot detects the ball directly in front of it, while it's close enough that the received signal strength meets a certain threshold, and facing towards the enemy goal line, the robot will charge at the while these 3 conditions are met. If the robot detects the ball directly in front of it but is at a distance, the robot will slowly approach the ball until the received signal strength meets a threshold. The robot approaches the ball at a slow pace to reach it without accidentally knocking the ball forward and perpetuating the loop condition. The robot is also programmed to orient itself towards the enemy goal line if it ever has the ball in position (assumed from a higher received signal strength and ball is detected straight ahead) while facing towards our goal line.

Charge and Dribble Strategies

The charge(), charge2(), and dribber() functions are essentially the same. They make the robot move forward at a motor speed of 2000 degrees/second. While they all use the run_time() function for the EV3 motors, what differentiates between them is how long they make the robot run for. They make the robot run charge forward from 200-500 ms.

Red Turn Strategy

The red_turn() is called whenever the robot's color sensor detects it moving across the enemy line (assumed to be the color red in this case). In general, whenever the robot detects itself moving across the red line, it backs up and turns around. The red_turn function decides which direction the robot should turn depending on the gyroscope reading. This is to prevent the robot from getting stuck on a wall if it decides to turn towards it.

Challenges Faced

Going into this project, we theorized the main challenges to be the robot's movement, and while they made up most of the work, the IRSeeker sensor was the main point of stress. The sensor is inconsistent with its accuracy and sometimes fails at more than 8in away or closer than 1in from the sensor. Similarly, regions 1,2,8 and 9 triggered the sensor, but the signal is so weak that the robot can get stuck in a loop of detecting the IR rays and immediately losing them. With the delay in entering the functions for the appropriate action, the robot frequently finds itself stuck, unable to progress unless the ball was moved into the other sensor zones.

Grabbing and keeping the ball became a bigger issue than anticipated as well. We tried many bumper designs, most of which were unable to keep the ball while turning, and others unable to effectively allow for a "kick." We settled on a narrower design. This bumper still lets the ball go if the turn is too fast, but allows for kicks as well, so the tradeoff was deemed acceptable.

The ball_find() function was always part of the original design concept, but the sensor's inconsistency led to the function being a bigger part of the robot's behavior than anticipated. The

main goal of this function is to find the ball, but as we tested the robot, we saw that an excessive amount of time was spent in this state, leading to many action cases being implemented. More time was put into this function than almost all the other parts in the robot's development cycle.

Surprisingly, the robot's movement was one of the lightest parts of its development process. In project 2 we used the run_time() motor function to move the robot. We soon realized that while running for the specified amount of time, it could not process any other sensor data, so we had to find another way to move the robot in a controlled manner. We implemented 'for' loops of an arbitrary length that allowed for the same functionality as the run_time() function but with the added effect of being able to pull sensor data at the same time, which solved the problem. The physical design, and improvements made to the code that moved the robot, allowed the robot to move smoothly and without issues in most cases.

References

IR Seeker Documentation: Appendix A: Sensor Data — ev3dev-stretch Linux kernel drivers 19 documentation