Comp 598EA Lego-mindstorms EV3

Musical Chairs

Spring 2015

Group Members: Jared Fowler, David Greenberg, Ilia Benson

Project Description/Parameters: Build a mobile device using the Lego-mindstorms EV3 educational kit. Program it to search for the center of a black annulus which is printed onto white paper. (Outer diameter: 8 inches. Inner diameter: 0.5 inches.) These annuluses will be placed in a playing field with a surrounding wall of 3.5 inches in height. The robot is to do this without any form of human intervention.

Physical Model: After several attempts to make a more complex model using multiple gears for mechanical advantage, dual-motor drive with single-motor steering, etc., we decided to settle for a very basic model, as can be seen in figures 1.1 – 1.2. This model was built completely custom, though, it does resemble the basic starting design provided by the lego-mindstorms' kit. Each large front wheel is mounted on its own motor. The smaller back wheels are elevated off the ground by about 1 centimeter, and are only there to help prevent the robot from being tipped over. The rear, and most of the weight, rests upon the metal track ball. This allows for in-place 360 degree rotations. The ultra-sonic sensor is mounted in the front at a height of about 3 inches. The color sensor is mounted about 1 centimeter off the ground, and about 1 inch from the center of the robot. Two touch sensors are mounted on the back. The total length and width of the robot are 7 and 5 inches respectively.

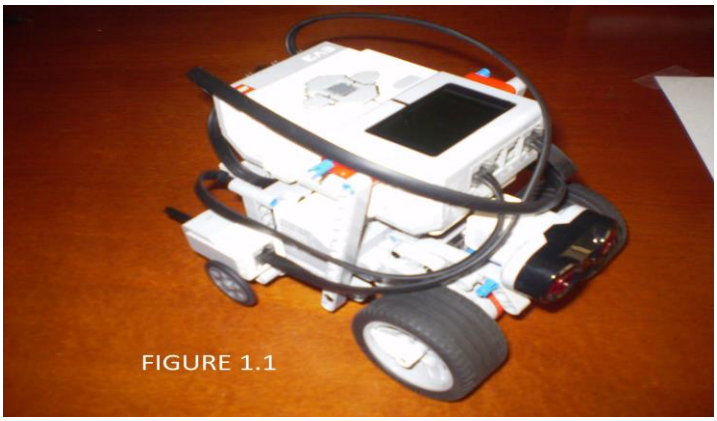

FIGURE 1.1

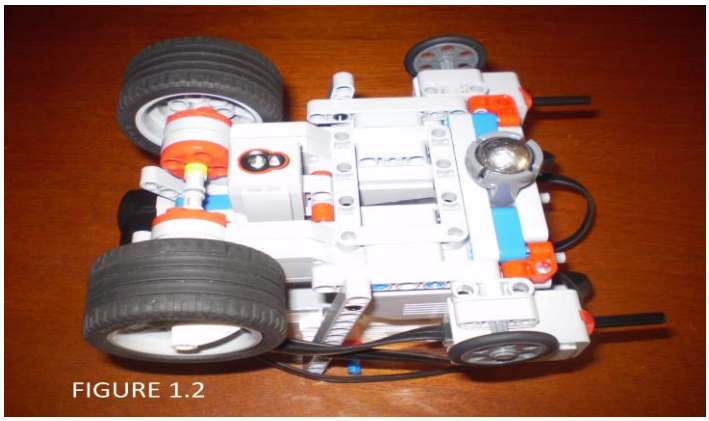

FIGURE 1.2

Programming Architecture: The device was programmed in Java, using the Eclipse IDE with the Lejos plug-in. Upon program startup, the device will first initialize various hardware and software components and will secondly enter the main loop. The main loop is status-flag driven, where a particular procedural function will be called based upon the robot's current status (Lost, on black, on white). The procedure which attempts to find the annulus can be labeled as a very simple round-robin. Sensors, in this procedure, are checked in this order: color, touch, ultra-sonic. A collection of constant variables reside at the top of the java class definition. These can be altered to fine-tune the behavior of the robot.

Functionality:

- **Sensor Initializations**: Each sensor is initialized and given its own sample buffer. (Telemetry taken from sensors are returned in a series of floats.) The color sensor's return value is configured to return a value 0-7 which represents an enumeration of colors. (None, Black, Blue, Green, Yellow, Red, White, Brown) The ultra-sonic

sensor's return value is configured to return the distance in meters (I've found its max range to be around 2.2 meters.), and finally, the touch sensors return the value 1 if deployed and 0 otherwise.

- **Motor Initializations:** Synchronization between the two motors is initialized and ready for use with the "startSynchronization" and "endSynchronization" `function calls.`
- **Movement:** Every function that commands the motors starts with startSynchronization and ends with endSynchonization. Before implementing this, the robot would jerk around when starting up and coming to a stop. This was specifically undesirable when fine-tune turning had to be done when the annulus was found. This hazard is avoided completely with the synchronization.
- **Audio:** Various sound clips were uploaded into the device and are played upon various events, such as finding the target for example. Because a call to the playSample function will suspend all other activity, the audio is played on a separate thread. It was quickly discovered that the device would freeze up if attempting to play more audio files than one at a time. (Perhaps having > 2 threads.) This was worked around by dedicating a thread to an on-going audio loop inside a nested class, which is flag-driven via flags set in functions. Unfortunately, the EV3 device seems to have persistent-random behavior and either works fine or crashes the system. Hours were spent trying to fix this without avail. The audio thread has been disabled. I should note that this makes the touch sensors useless.
- **Lost Procedure:** Upon program startup, the robot's status is set as "LOST", hence, the lost procedure is called. The procedure is simple: Move forward until we find the color black on the ground, (Originally it was programmed to also look for the white paper which surround the black annulus, but we felt that there might not be a distinction made between the floor color and the paper.) or, if an obstacle is detected in front of us, rotate at a certain angle, where 90 < angle < 180, and start over. If one of the touch sensors are activated, play some audio. Just for fun, an audio clip is also set to play after an x-amount of time has passed and the robot is still in the "LOST" status.
- **Annulus Procedure:** Once an annulus is found, or a black surface that we mistakenly assume to be an annulus, the status flag is changed and this procedure begins. This procedure is nothing like the "LOST" procedure which is more of a round-robin approach. This procedure is a one-time pass through. It is SMART, but also very VULNERABLE. It essentially works like this: Upon entering the procedure the robot has come to a complete stop and is somewhere on/near the black. If we skimmed the edge and are not on black, try to get back onto it. If we fail to do so, return to "LOST" status. If we are on black, back up until we hit the non-black. Move over the black surface in increments until we hit another non-black surface. Determine if the surface is either outside the annulus or the center white circle. If found to be the outer-non-black surface, back up half-way based upon the stored number of increments, turn 90 degrees, and based upon what we know about right triangles, calculate the distance needed to travel. (See figure 2.1) Travel that distance one way, and if necessary, back up twice the distance if we went the wrong way. Of course, this is a simplistic view of it. The code also takes into account the axis of rotation in respect to the center of the robot, location of color sensor in respect to the center of the robot, etc., all of which slightly modify the procedure.

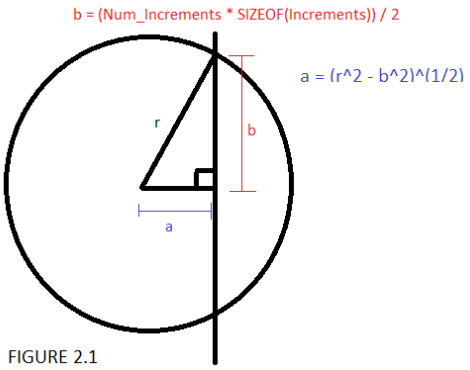

FIGURE 2.1

Summary/Conclusion: If left by itself, the robot is very effective at completing its mission, especially upon reaching the annulus. Its smart maneuvering, however, hasn't taken into much consideration what would happen if another robot was to collide with it and throw it off course. Probably not a problem in the "LOST" stage, but in the non-"LOST" stage, it could have significant consequences. Fun project!